

A Brain-Friendly Guide

2nd Edition
Updated for HTML5

Head First HTML and CSS

Launch your
web career in
one chapter



A learner's guide
to creating
standards-based
web pages



Watch out for
common HTML & CSS
traps and pitfalls

Bend your mind
around 100 puzzles
& exercises



Learn why everything
your friends know about
style is probably wrong

Avoid
embarrassing
validation mistakes



O'REILLY®

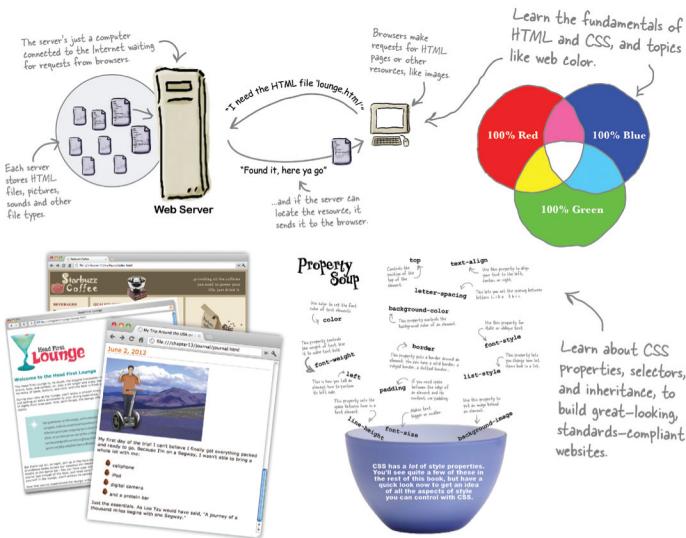
Elisabeth Robson & Eric Freeman

Head First HTML and CSS

Web Design & Development/HTML

What will you learn from this book?

Tired of reading HTML books that only make sense after you're an expert? Then it's about time you picked up the newly revised *Head First HTML and CSS* and really learned HTML. You want to learn HTML and CSS so you can finally create those web pages you've always wanted, so you can communicate more effectively with friends, family, fans, and fanatic customers. You also want to do it right, using the latest HTML5 standards, so you can actually maintain and expand your web pages over time so they work in all browsers and mobile devices.



What's so special about this book?

In this book, you'll learn the real secrets of creating web pages, and, most importantly, you'll learn them in a way that won't put you to sleep. If you've read a Head First book, you know what to expect: a visually rich format designed for the way your brain works. Using the latest research in neurobiology, cognitive science, and learning theory, this book will load HTML and CSS into your brain in a way that sticks.

US \$39.99

CAN \$41.99

ISBN: 978-0-596-15990-0



twitter.com/headfirstlabs
facebook.com/HeadFirst

O'REILLY®
oreilly.com
headfirstlabs.com

"Head First HTML and CSS is a thoroughly modern introduction to forward-looking practices in web page markup and presentation. It correctly anticipates readers' puzzlements and handles them just in time. The highly graphic and incremental approach precisely mimics the best way to learn this stuff: make a small change and see it in the browser to understand what each new item means."

— Danny Goodman,
author of *Dynamic HTML: The Definitive Guide*

"The information covered in this book is the same material the pros know, but taught in an educational and humorous manner that doesn't ever make you think the material is impossible to learn or you are out of your element."

— Christopher Schmitt,
author of *The CSS Cookbook* and *Professional CSS*

Praise for *Head First HTML and CSS*

“*Head First HTML and CSS* is a thoroughly modern introduction to forward-looking practices in web page markup and presentation. It correctly anticipates readers’ puzzlements and handles them just in time. The highly graphic and incremental approach precisely mimics the best way to learn this stuff: make a small change and see it in the browser to understand what each new item means.”

— **Danny Goodman, author of *Dynamic HTML: The Definitive Guide***

“Eric Freeman and Elisabeth Robson clearly know their stuff. As the Internet becomes more complex, inspired construction of web pages becomes increasingly critical. Elegant design is at the core of every chapter here, each concept conveyed with equal doses of pragmatism and wit.”

— **Ken Goldstein, Executive Vice President and Managing Director, Disney Online**

“The Web would be a much better place if every HTML author started off by reading this book.”

— **L. David Baron, Technical Lead, Layout and CSS, Mozilla Corporation**
<http://dbaron.org/>

“I’ve been writing HTML and CSS for 10 years now, and what used to be a long trial-and-error learning process has now been reduced neatly into an engaging paperback. HTML used to be something you could just hack away at until things looked okay on screen, but with the advent of web standards and the movement toward accessibility, sloppy coding practice is not acceptable anymore...from a business standpoint or a social responsibility standpoint. *Head First HTML and CSS* teaches you how to do things right from the beginning without making the whole process seem overwhelming. HTML, when properly explained, is no more complicated than plain English, and the authors do an excellent job of keeping every concept at eye level.”

— **Mike Davidson, President and CEO, Newsvine, Inc.**

“The information covered in this book is the same material the pros know, but taught in an educational and humorous manner that doesn’t ever make you think the material is impossible to learn or you are out of your element.”

— **Christopher Schmitt, author of *The CSS Cookbook* and *Professional CSS*, schmitt@christopher.org**

“Oh, great. You made an HTML book simple enough a CEO can understand it. What will you do next? Accounting simple enough my developer can understand it? Next thing you know, we’ll be collaborating as a team or something.”

— **Janice Fraser, CEO, Adaptive Path**

More Praise for *Head First HTML and CSS*

“I *heart* *Head First HTML and CSS*—it teaches you everything you need to learn in a ‘fun coated’ format!”

— **Sally Applin, UI designer and fine artist, <http://sally.com>**

“This book has humor and charm, but most importantly, it has heart. I know that sounds ridiculous to say about a technical book, but I really sense that at its core, this book (or at least its authors) really care that the reader learns the material. This comes across in the style, the language, and the techniques. Learning—real understanding and comprehension—on the part of the reader is clearly topmost in the minds of the authors. And thank you, thank you, thank you, for the book’s strong and sensible advocacy of standards compliance. It’s great to see an entry-level book, that I think will be widely read and studied, campaign so eloquently and persuasively on behalf of the value of standards compliance in web page code. I even found in here a few great arguments I had not thought of—ones I can remember and use when I am asked (as I still am)—‘what’s the deal with compliance and why should we care?’ I’ll have more ammo now! I also liked that the book sprinkles in some basics about the mechanics of actually getting a web page live—FTP, web server basics, file structures, etc.”

— **Robert Neer, Director of Product Development, Movies.com**

“*Head First HTML and CSS* is a most entertaining book for learning how to build a great web page. It not only covers everything you need to know about HTML and CSS, it also excels in explaining everything in layman’s terms with a lot of great examples. I found the book truly enjoyable to read, and I learned something new!”

— **Newton Lee, Editor-in-Chief, ACM Computers in Entertainment
<http://www.acmcie.org>**

“My wife stole the book. She’s never done any web design, so she needed a book like *Head First HTML and CSS* to take her from beginning to end. She now has a list of websites she wants to build—for our son’s class, our family...If I’m lucky, I’ll get the book back when she’s done.”

— **David Kaminsky, Master Inventor, IBM**

“Beware. If you’re someone who reads at night before falling asleep, you’ll have to restrict *Head First HTML and CSS* to daytime reading. This book wakes up your brain.”

— **Pauline McNamara, Center for New Technologies and Education,
Fribourg University, Switzerland**

Praise for other books by Eric Freeman and Elisabeth Robson

“From the awesome *Head First Java* folks, this book uses every conceivable trick to help you understand and remember. Not just loads of pictures: pictures of humans, which tend to interest other humans. Surprises everywhere. Stories, because humans love narrative. (Stories about things like pizza and chocolate. Need we say more?) Plus, it’s darned funny.”

— **Bill Camarda, READ ONLY**

“This book’s admirable clarity, humor, and substantial doses of clever make it the sort of book that helps even nonprogrammers think well about problem solving.”

— **Cory Doctorow, co-editor of Boing Boing
and author of Down and Out in the Magic Kingdom
and Someone Comes to Town, Someone Leaves Town**

“I feel like a thousand pounds of books have just been lifted off of my head.”

— **Ward Cunningham, inventor of the wiki
and founder of the Hillside Group**

“This book is close to perfect, because of the way it combines expertise and readability. It speaks with authority and it reads beautifully. It’s one of the very few software books I’ve ever read that strikes me as indispensable. (I’d put maybe 10 books in this category, at the outside.)”

— **David Gelernter, professor of computer science,
Yale University, and author of Mirror Worlds and Machine Beauty**

“A nosedive into the realm of patterns, a land where complex things become simple, but where simple things can also become complex. I can think of no better tour guides than these authors.”

— **Miko Matsumura, industry analyst, The Middleware Company
former Chief Java Evangelist, Sun Microsystems**

“I laughed, I cried, it moved me.”

— **Daniel Steinberg, Editor-in-Chief, java.net**

“Just the right tone for the geeked-out, casual-cool guru coder in all of us. The right reference for practical development strategies—gets my brain going without having to slog through a bunch of tired, stale professor-speak.”

— **Travis Kalanick, founder of Scour and Red Swoosh,
member of the MIT TR100**

“I literally love this book. In fact, I kissed this book in front of my wife.”

— **Satish Kumar**

Other O'Reilly books by Eric Freeman and Elisabeth Robson

Head First Design Patterns

Head First HTML with CSS & XHTML (first edition)

Head First HTML5 Programming

Other related books from O'Reilly

HTML5: Up and Running

HTML5 Canvas

HTML5: The Missing Manual

HTML5 Geolocation

HTML5 Graphics with SVG and CSS3

HTML5 Forms

HTML5 Media

Other books in O'Reilly's *Head First* series

Head First C#

Head First Java

Head First Object-Oriented Analysis & Design (OOA&D)

Head First Servlets and JSP

Head First SQL

Head First Software Development

Head First JavaScript

Head First Ajax

Head First Rails

Head First PHP & MySQL

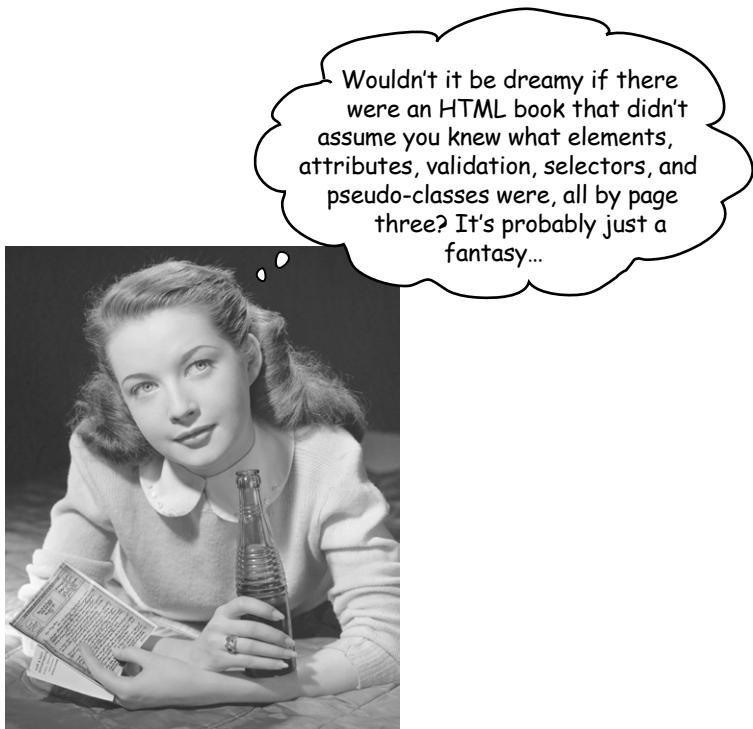
Head First Web Design

Head First Networking

Head First iPhone and iPad Development

Head First jQuery

Head First HTML and CSS



Elisabeth Robson
Eric Freeman

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Head First HTML and CSS

by Elisabeth Robson and Eric Freeman

Copyright © 2012 Elisabeth Robson and Eric Freeman. All rights reserved.

Printed in Canada.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://my.safaribooksonline.com>). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Series Creators: Kathy Sierra, Bert Bates

Editor: Brett McLaughlin (first edition), Mike Hendrickson (second edition)

Cover Designer: Karen Montgomery

HTML Wranglers: Elisabeth Robson, Eric Freeman

Production Editor: Kristen Borg

Indexer: Ron Strauss

Proofreader: Rachel Monaghan

Page Viewer: Oliver



Printing History:

December 2005: First Edition.

September 2012: Second Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. The *Head First* series designations, *Head First HTML and CSS*, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and the authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

In other words, if you use anything in *Head First HTML and CSS* to, say, run a nuclear power plant, you're on your own. We do, however, encourage you to visit the Head First Lounge.

No elements or properties were harmed in the making of this book.

Thanks to Clemens Orth for the use of his photo, "applestore.jpg", which appears in Chapter 5.

ISBN: 978-0-596-15990-0

[TI]

Browser wars? You'll
find out in Chapter 6.

To the W3C, for saving us from the browser wars and
for their brilliance in separating structure (HTML) from
presentation (CSS)...

And for making HTML and CSS complex enough that
people need a book to learn it.

Authors of Head First HTML and CSS



Eric is described by Head First series co-creator Kathy Sierra as “one of those rare individuals fluent in the language, practice, and culture of multiple domains from hipster hacker, corporate VP, engineer, think tank.”

Professionally, Eric recently ended nearly a decade as a media company executive—having held the position of CTO of Disney Online and Disney.com at the Walt Disney Company. Eric is now devoting his time to WickedlySmart, a startup he co-created with Elisabeth.

By training, Eric is a computer scientist, having studied with industry luminary David Gelernter during his Ph.D. work at Yale University. His dissertation is credited as the seminal work in alternatives to the desktop metaphor, and also as the first implementation of activity streams, a concept he and Dr. Gelernter developed.

In his spare time, Eric is deeply involved with music; you’ll find Eric’s latest project, a collaboration with ambient music pioneer Steve Roach, available on the iPhone App Store under the name Immersion Station.

Eric lives with his wife and young daughter on Bainbridge Island. His daughter is a frequent visitor to Eric’s studio, where she loves to turn the knobs of his synths and audio effects.

Write to Eric at eric@wickedlysmart.com or visit his site at <http://ericfreeman.com>.

↙ Elisabeth Robson



Elisabeth is a software engineer, writer, and trainer. She has been passionate about technology since her days as a student at Yale University, where she earned a master’s of science in computer science and designed a concurrent, visual programming language and software architecture.

Elisabeth’s been involved with the Internet since the early days; she co-created the award-winning website, the Ada Project, one of the first websites designed to help women in computer science find career and mentorship information online.

She’s currently co-founder of WickedlySmart, an online education experience centered on web technologies, where she creates books, articles, videos and more. Previously, as Director of Special Projects at O’Reilly Media, Elisabeth produced in-person workshops and online courses on a variety of technical topics and developed her passion for creating learning experiences to help people understand technology. Prior to her work with O’Reilly, Elisabeth spent time spreading fairy dust at the Walt Disney Company, where she led research and development efforts in digital media.

When not in front of her computer, you’ll find Elisabeth hiking, cycling or kayaking in the great outdoors, with her camera nearby, or cooking vegetarian meals.

You can send her email at beth@wickedlysmart.com or visit her blog at <http://elisabethrobson.com>.

Table of Contents (summary)

	Intro	xxv
1	The Language of the Web: <i>getting to know html</i>	1
2	Meet the “HT” in HTML: <i>going further, with hypertext</i>	43
3	Web Page Construction: <i>building blocks</i>	77
4	A Trip to Webville: <i>getting connected</i>	123
5	Meeting the Media: <i>adding images to your pages</i>	163
6	Serious HTML: <i>standards and all that jazz</i>	219
7	Adding a Little Style: <i>getting started with CSS</i>	255
8	Expanding your Vocabulary: <i>styling with fonts and colors</i>	311
9	Getting Intimate with Elements: <i>the box model</i>	361
10	Advanced Web Construction: <i>divs and spans</i>	413
11	Arranging Elements: <i>layout and positioning</i>	471
12	Modern HTML: <i>html5 markup</i>	545
13	Getting Tabular: <i>tables and more lists</i>	601
14	Getting Interactive: <i>html forms</i>	645
	Appendix: The Top Ten Topics (We Didn’t Cover): <i>leftovers</i>	697

Table of Contents (the real thing)

Intro

Your brain on HTML and CSS. Here you are trying to *learn* something, while here your *brain* is doing you a favor by making sure the learning doesn’t *stick*. Your brain’s thinking, “Better leave room for more important things, like which wild animals to avoid and whether naked snowboarding is a bad idea.” So how *do* you trick your brain into thinking that your life depends on knowing HTML and CSS?

Who is this book for?	xxvi
Metacognition	xxix
Here’s what WE did	xxx
Bend your brain into submission	xxxi
Tech reviewers (first edition)	xxxiv
Acknowledgments (first edition)	xxxv
Tech reviewers (second edition)	xxxvi
Acknowledgments (second edition)	xxxvi

getting to know html

1

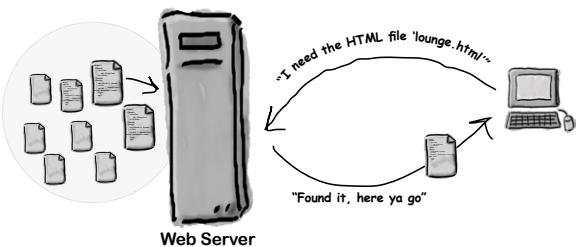
The Language of the Web

The only thing that is standing between you and getting yourself on the Web is learning to speak the lingo:

HyperText Markup Language, or HTML for short. So, get ready for some language lessons. After this chapter, not only are you going to understand some basic **elements** of HTML, but you'll also be able to speak HTML with a little **style**. Heck, by the end of this book, you'll be talking HTML like you grew up in Webville.



The web killed the radio star	2
What does the web server do?	3
What you write (the HTML)	4
What the browser creates	5
Your big break at Starbuzz Coffee	9
Creating the Starbuzz web page	11
Creating an HTML file (Mac)	12
Creating an HTML file (Windows)	14
Meanwhile, back at Starbuzz Coffee...	17
Saving your work	18
Opening your web page in a browser	19
Take your page for a test drive	20
Are we there yet?	23
Another test drive	24
Tags dissected	25
Meet the style element	29
Giving Starbuzz some style...	30
Cruisin' with style...	31
Exercise Solutions	38



going further with hypertext

2

Meeting the “HT” in HTML

Did someone say “hypertext?” What’s that? Oh, only the entire basis of the Web. In Chapter 1 we kicked the tires of HTML and found it to be a nice markup language (the “ML” in HTML) for describing the structure of web pages. Now we’re going to check out the “HT” in HTML, hypertext, which will let us break free of a single page and link to other pages. Along the way we’re going to meet a powerful new element, the `<a>` element, and learn how being “relative” is a groovy thing. So, fasten your seat belts—you’re about to learn some hypertext.



Head First Lounge, <i>new and improved</i>	44
Creating the new lounge	46
What did we do?	48
Understanding attributes	51
Getting organized	56
Organizing the lounge...	57
Technical difficulties	58
Planning your paths...	60
Fixing those broken images...	66
Exercise Solutions	73



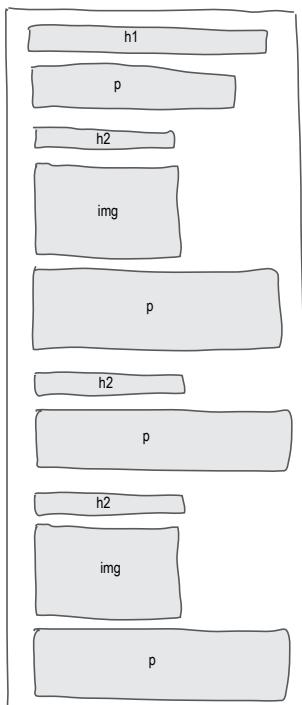
building blocks

3

Web Page Construction

I was told I'd actually be creating web pages in this book?

You've certainly learned a lot already: tags, elements, links, paths...but it's all for nothing if you don't create some killer web pages with that knowledge. In this chapter we're going to ramp up construction: you're going to take a web page from conception to blueprint, pour the foundation, build it, and even put on some finishing touches. All you need is your hard hat and your toolbelt, as we'll be adding some new tools and giving you some insider knowledge that would make Tim "The Toolman" Taylor proud.



From journal to website, at 12 mph	79
The rough design sketch	80
From a sketch to an outline	81
From the outline to a web page	82
Test-driving Tony's web page	84
Adding some new elements	85
Meet the <q> element	86
Looooong quotes	90
Adding a blockquote	91
The real truth behind the <q> and <blockquote> mystery	94
Meanwhile, back at Tony's site...	100
Of course, you could use the <p> element to make a list...	101
Constructing HTML lists in two easy steps	102
Taking a test drive through the cities	104
Putting one element inside another is called "nesting"	107
To understand the nesting relationships, draw a picture	108
Using nesting to make sure your tags match	109
Exercise Solutions	117

getting connected

4

A Trip to Webville

Web pages are a dish best served on the Internet. So far you've only created HTML pages that live on your own computer. You've also only linked to pages that are on your own computer. We're about to change all that. In this chapter we'll encourage you to get those web pages on the Internet where all your friends, fans, and customers can actually see them. We'll also reveal the mysteries of linking to other pages by cracking the code of the h, t, t, p, ., /, /, w, w, w. So, gather your belongings; our next stop is Webville.

Getting Starbuzz (or yourself) onto the Web	124
Finding a hosting company	125
How can you get a domain name?	126
Moving in	128
Getting your files to the root folder	129
As much FTP as you can possibly fit in two pages	130
Back to business...	133
Mainstreet, USA	134
What is HTTP?	135
What's an absolute path?	136
How default pages work	139
Earl needs a little help with his URLs	140
How do we link to other websites?	142
Linking to Caffeine Buzz	143
And now for the test drive...	144
Web page fit and finish	147
The title test drive...	148
Linking into a page	149
Using the id attribute to create a destination for <a>	150
How to link to elements with ids	151
Linking to a new window	155
Opening a new window using target	156
Exercise Solutions	160

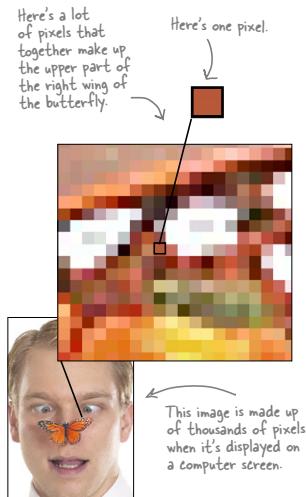


adding images to your pages

5

Meeting the Media

Smile and say “cheese.” Actually, smile and say “gif,” “jpg,” or “png”—these are going to be your choices when “developing pictures” for the Web. In this chapter you’re going to learn all about adding your first media type to your pages: images. Got some digital photos you need to get online? No problem. Got a logo you need to get on your page? Got it covered. But before we get into all that, don’t you still need to be formally introduced to the `` element? So sorry, we weren’t being rude; we just never saw the “right opening.” To make up for it, here’s an entire chapter devoted to ``. By the end of the chapter you’re going to know all the ins and outs of how to use the `` element and its attributes. You’re also going to see exactly how this little element causes the browser to do extra work to retrieve and display your images.



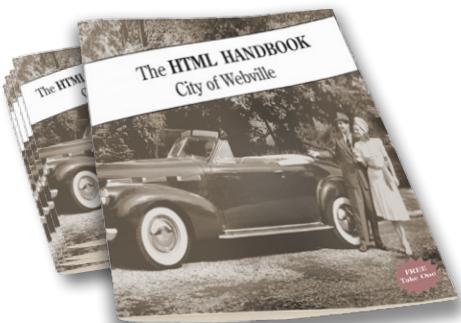
How the browser works with images	164
How images work	167
<code></code> : it's not just relative links anymore	171
Always provide an alternative	173
Sizing up your images	174
Creating the ultimate fan site: myPod	175
Whoa! The image is way too large	178
Open the image	182
Resizing the image	183
Fixing up the myPod HTML	188
More photos for myPod	190
Turning the thumbnails into links	196
Create individual pages for the photos	197
So, how do I make links out of images?	198
What format should we use?	203
To be transparent, or not to be transparent? That is the question...	204
Wait, what is the color of the web page background?	206
Check out the logo with a matte	207
Add the logo to the myPod web page	208
Exercise Solutions	213

standards and all that jazz

6

Serious HTML

What else is there to know about HTML? You're well on your way to mastering HTML. In fact, isn't it about time we move on to CSS and learn how to make all this bland markup look fabulous? Before we do, we need to make sure your HTML is really ready for the big leagues. Don't get us wrong, you've been writing first-class HTML all along, but there are just a few extra things you need to do to make it "industry standard" HTML. It's also time you think about making sure you're using the latest and greatest HTML standard, otherwise known as HTML5. By doing so, you'll ensure that your pages play well with the latest i-Device, and that they'll display more uniformly across all browsers (at least the ones you'd care about). You'll also have pages that load faster, pages that are guaranteed to play well with CSS, and pages that are ready to move into the future as the standards grow. Get ready, this is the chapter where you move from web tinkerer to web professional.



A Brief History of HTML	222
The new, and improved, HTML5 doctype	227
HTML, the new "living standard"	228
Adding the document type definition	229
The doctype test drive	230
Meet the W3C validator	233
Validating the Head First Lounge	234
Houston, we have a problem...	235
Fixing that error	236
We're almost there...	237
Adding a <meta> tag to specify the character encoding	239
Making the validator (and more than a few browsers) happy with a <meta> tag...	240
Third time's the charm?	241
Calling all HTML professionals, grab the handbook...	244
Exercise Solutions	251

getting started with CSS

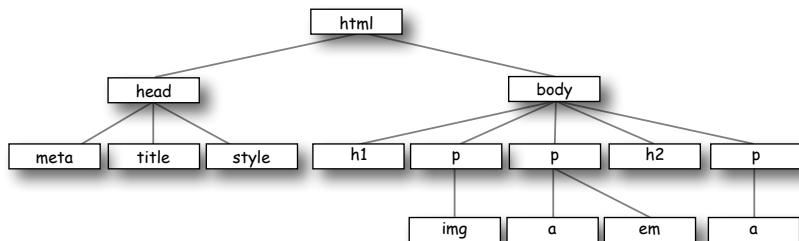
7

Adding a Little Style

I was told there'd be CSS in this book. So far you've been concentrating on learning HTML to create the structure of your web pages. But as you can see, the browser's idea of style leaves a lot to be desired. Sure, we could call the fashion police, but we don't need to. With CSS, you're going to completely control the presentation of your pages, often without even changing your HTML. Could it really be so easy? Well, you *are* going to have to learn a new language; after all, Webville is a bilingual town. After reading this chapter's guide to learning the language of CSS, you're going to be able to stand on *either* side of Main Street and hold a conversation.



You're not in Kansas anymore	256
Overheard on Webville's "Trading Spaces"	258
Using CSS with HTML	259
Getting CSS into your HTML	261
Adding style to the lounge	262
Let's put a line under the welcome message too	265
So, how do selectors really work?	267
Seeing selectors visually	270
Getting the Lounge style into the elixirs and directions pages	273
It's time to talk about your inheritance...	281
Overriding inheritance	284
Adding an element to the greentea class	287
Creating a class selector	288
Taking classes further...	290
The world's smallest and fastest guide to how styles are applied	292
Exercise Solutions	303



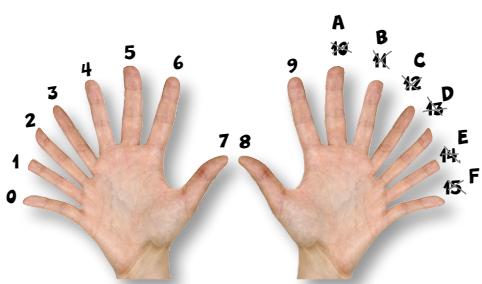
styling with fonts and colors

8

Expanding Your Vocabulary

Your CSS language lessons are coming along nicely. You already have the basics of CSS down, and you know how to create CSS rules to select and specify the style of an element. Now it's time to build your vocabulary, and that means picking up some new properties and learning what they can do for you. In this chapter we're going to work through some of the most common properties that affect the display of text. To do that, you'll need to learn a few things about fonts and color. You're going to see you don't have to be stuck with the fonts everyone else uses, or the clunky sizes and styles the browser uses as the defaults for paragraphs and headings. You're also going to see there is a lot more to color than meets the eye.

Text and fonts from 30,000 feet	312
What is a font family anyway?	314
Specifying font families using CSS	317
Dusting off Tony's journal	318
How do I deal with everyone having different fonts?	321
How Web Fonts work	323
How to add a Web Font to your page...	325
Adjusting font sizes	328
So, how should I specify my font sizes?	330
Let's make these changes to the font sizes in Tony's web page	332
Changing a font's weight	335
Adding style to your fonts	337
Styling Tony's quotes with a little italic	338
How do web colors work?	340
How do I specify web colors? Let me count the ways...	343
The two-minute guide to hex codes	346
How to find web colors	348
Back to Tony's page...	351
Everything you ever wanted to know about text-decorations	353
Removing the underline...	354
Exercise Solutions	357



9

the box model

Getting Intimate with Elements

To do advanced web construction, you really need to know your building materials. In this chapter we're going to take a close look at our building materials: the HTML elements. We're going to put block and inline elements right under the microscope and see what they're made of. You'll see how you can control just about every aspect of how an element is constructed with CSS. But we don't stop there—you'll also see how you can give elements unique identities. And, if that weren't enough, you're going to learn when and why you might want to use multiple stylesheets. So, turn the page and start getting intimate with elements.

The lounge gets an upgrade	362
Starting with a few simple upgrades	364
Checking out the new line height	366
Getting ready for some major renovations	367
A closer look at the box model	368
What you can do to boxes	370
Creating the guarantee style	375
A test drive of the paragraph border	376
Padding, border, and margins for the guarantee	377
Adding a background image	380
Fixing the background image	383
How do you add padding only on the left?	384
How do you increase the margin just on the right?	385
A two-minute guide to borders	386
Border fit and finish	389
Using an id in the lounge	396
Using multiple stylesheets	399
Stylesheets—they're not just for desktop browsers anymore...	400
Add media queries right into your CSS	401
Exercise Solutions	407



divs and spans

10



Advanced Web Construction

It's time to get ready for heavy construction. In this chapter we're going to roll out two new HTML elements: <div> and . These are no simple "two by fours"; these are full-blown steel beams. With <div> and , you're going to build some serious supporting structures, and once you've got those structures in place, you're going to be able to style them all in new and powerful ways. Now, we couldn't help but notice that your CSS toolbelt is really starting to fill up, so it's time to show you a few shortcuts that will make specifying all these properties a lot easier. And we've also got some special guests in this chapter, the *pseudo-classes*, which are going to allow you to create some very interesting selectors. (If you're thinking that "pseudo-classes" would make a great name for your next band, too late; we beat you to it.)

A close look at the elixirs HTML	415
Let's explore how we can divide a page into logical sections	417
Adding a border	424
Adding some real style to the elixirs section	425
Working on the elixir width	426
Adding the basic styles to the elixirs	431
What we need is a way to select descendants	437
Changing the color of the elixir headings	439
Fixing the line height	440
It's time to take a little shortcut	442
Adding s in three easy steps	448
The <a> element and its multiple personalities	452
How can you style elements based on their state?	453
Putting those pseudo-classes to work	455
Isn't it about time we talk about the "cascade"?	457
The cascade	459
Welcome to the "What's my specificity?" game	460
Putting it all together	461
Exercise Solutions	467

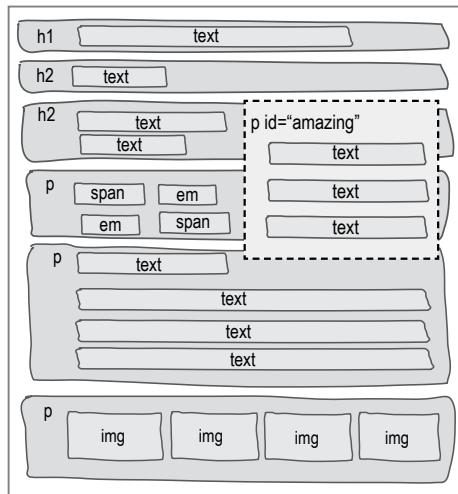
11

layout and positioning

Arranging Elements



It's time to teach your HTML elements new tricks. We're not going to let those HTML elements just sit there anymore—it's about time they get up and help us create some pages with real *layouts*. How? Well, you've got a good feel for the `<div>` and `` structural elements and you know all about how the box model works, right? So, now it's time to use all that knowledge to craft some real designs. No, we're not just talking about more background and font colors—we're talking about full-blown professional designs using multicolumn layouts. This is the chapter where everything you've learned comes together.



Did you do the Super Brain Power?	472
Use the Flow, Luke	473
What about inline elements?	475
How it all works together	476
How to float an element	479
The new Starbuzz	483
Move the sidebar just below the header	488
Fixing the two-column problem	491
Setting the margin on the main section	492
Solving the overlap problem	495
Righty tighty, lefty loosey	498
Liquid and frozen designs	501
How absolute positioning works	504
Changing the Starbuzz CSS	507
How CSS table display works	511
Adding HTML structure for the table display	513
What's the problem with the spacing?	517
Problems with the header	524
Fixing the header images with float	525
Positioning the award	528
How does fixed positioning work?	531
Using a negative left property value	533
Exercise Solutions	539

html5 markup

12

Modern HTML

So, we're sure you've heard the hype around HTML5. And, given how far along you are in this book, you're probably wondering if you made the right purchase. Now, one thing to be clear about, up front, is that everything you've learned in this book has been HTML, and more specifically has met the HTML5 standard. But there are some new aspects of HTML markup that were added with the HTML5 standard that we haven't covered yet, and that's what we're going to do in this chapter. Most of these additions are evolutionary, and you're going to find you are quite comfortable with them given all the hard work you've already done in this book. There's some revolutionary stuff too (like video), and we'll talk about that in this chapter as well. So, let's dive in and take a look at these new additions!



Rethinking HTML structure	546
Update your Starbuzz HTML	551
How to update your CSS for the new elements	554
Setting up the CSS for the blog page	563
We still need to add a date to the blog...	565
Adding the <time> element to your blog	566
How to add more <header> elements	568
So, what's wrong with the header anyway?	570
A final test drive for the headers	571
Completing the navigation	574
Who needs GPS? Giving the navigation a test drive	575
Ta-da! Look at that navigation!	577
Creating the new blog entry	580
Lights, camera, action...	581
How does the <video> element work?	583
Closely inspecting the video attributes...	584
What you need to know about video formats	586
The video format contenders	587
How to juggle all those formats...	589
How to be even more specific with your video formats	590
Exercise Solutions	597

13

tables and more lists

Getting Tabular

If it walks like a table and talks like a table... There comes a time in life when we have to deal with the dreaded *tabular data*. Whether you need to create a page representing your company's inventory over the last year or a catalog of your vinylmation collection (don't worry, we won't tell), you know you need to do it in HTML, but how? Well, have we got a deal for you: order now, and in a single chapter we'll reveal the secrets that will allow you to put your very own data right inside HTML tables. But there's more: with every order we'll throw in our exclusive guide to styling HTML tables. And, if you act now, as a special bonus, we'll throw in our guide to styling HTML lists. Don't hesitate; call now!

How do you make tables with HTML?	603
Creating a table with HTML	604
What the browser creates	605
Tables dissected	606
Adding a caption	609
Before we start styling, let's get the table into Tony's page	611
Getting those borders to collapse	616
How about some color?	618
Tony made an interesting discovery	620
Another look at Tony's table	621
How to tell cells to span more than one row	622
Test drive the table	624
Trouble in paradise?	625
Overriding the CSS for the nested table headings	629
Giving Tony's site the final polish	630
What if you want a custom marker?	632
Exercise Solutions	636



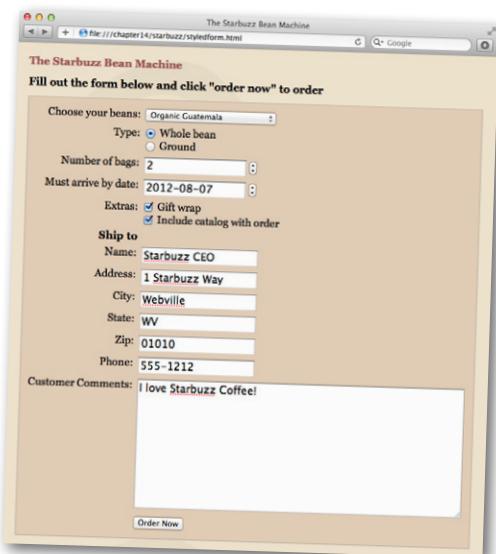
City	Date	Temperature	Altitude	Population	Diner Rating				
Walla Walla, WA	June 15th	75	1,204 ft	29,686	4/5				
Magic City, ID	June 25th	74	5,312 ft	50	3/5				
Bountiful, UT	July 10th	91	4,226 ft	41,173	4/5				
Last Chance, CO	July 23rd	102	4,780 ft	265	3/5				
	August 9th	93			5/5				
Truth or Consequences, NM	August 27th	98	4,242 ft	7,289	<table border="1"> <tr> <td>Tess</td> <td>5/5</td> </tr> <tr> <td>Tony</td> <td>4/5</td> </tr> </table>	Tess	5/5	Tony	4/5
Tess	5/5								
Tony	4/5								
Why, AZ	August 18th	104	860 ft	480	3/5				

14

html forms

Getting Interactive

So far all your web communication has been one-way: from your page to your visitors. Golly, wouldn't it be nice if your visitors could talk back? That's where HTML forms come in: once you enable your pages with forms (along with a little help from a web server), your pages are going to be able to gather customer feedback, take an online order, get the next move in an online game, or collect the votes in a "hot or not" contest. In this chapter you're going to meet a whole team of HTML elements that work together to create web forms. You'll also learn a bit about what goes on behind the scenes in the server to support forms, and we'll even talk about keeping those forms stylish.



How forms work	646
What you write in HTML	648
What the browser creates	649
How the <form> element works	650
Getting ready to build the Bean Machine form	660
Adding the <form> element	661
How form element names work	662
Back to getting those <input> elements into your HTML	664
Adding some more input elements to your form	665
Adding the <select> element	666
Give the customer a choice of whole or ground beans	668
Punching the radio buttons	669
Using more input types	670
Adding the number and date input types	671
Completing the form	672
Adding the checkboxes and text area	673
Watching GET in action	679
Getting the form elements into HTML structure	684
Styling the form with CSS	686
A word about accessibility	688
What more could possibly go into a form?	689
Exercise Solutions	693

15

appendix: leftovers

The Top Ten Topics (We Didn't Cover)

We covered a lot of ground, and you're almost finished with this book. We'll miss you, but before we let you go, we wouldn't feel right about sending you out into the world without a little more preparation. We can't possibly fit everything you'll need to know into this relatively short chapter. Actually, we *did* originally include everything you need to know about HTML and CSS (not already covered by the other chapters), by reducing the type point size to .00004. It all fit, but nobody could read it. So, we threw most of it away, and kept the best bits for this Top Ten appendix.



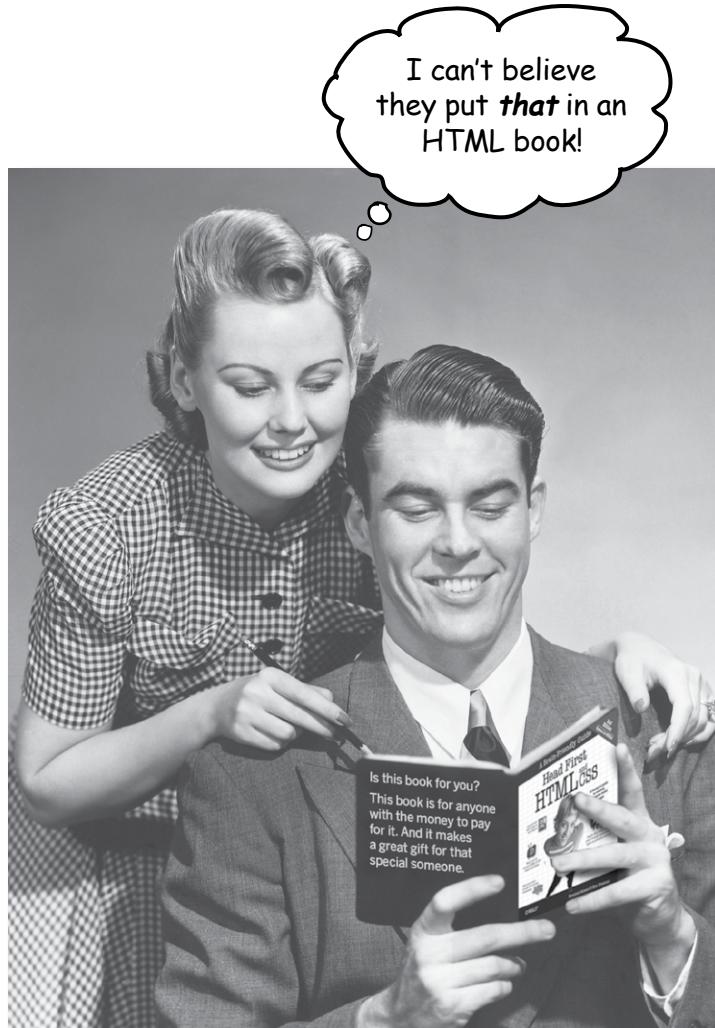
#1 More CSS selectors	698
#2 Vendor-specific CSS properties	700
#3 CSS transforms and transitions	701
#4 Interactivity	703
#5 HTML5 APIs and web apps	704
#6 More on Web Fonts	706
#7 Tools for creating web pages	707
#8 XHTML5	708
#9 Server-side scripting	709
#10 Audio	710

i Index

711

how to use this book

Intro



In this section, we answer the burning question:
"So, why DID they put that in an HTML book?"

Who is this book for?

If you can answer “yes” to all of these:

- ① Do you have access to a computer with a **web browser** and a **text editor**?
- ② Do you want to **learn, understand, and remember** how to **create** web pages using the best techniques and the most recent standards?
- ③ Do you prefer **stimulating dinner-party conversation** to **dry, dull, academic lectures**?

this book is for you.



If you have access to any computer manufactured in the last decade, the answer is yes.

Who should probably back away from this book?

If you can answer “yes” to any one of these:

- ① Are you **completely new to computers**?
(You don't need to be advanced, but you should understand folders and files, simple text editing applications, and how to use a web browser.)
- ② Are you a kick-butt web developer looking for a **reference book**?
- ③ Are you **afraid to try something different**? Would you rather have a root canal than mix stripes with plaid? Do you believe that a technical book can't be serious if HTML tags are anthropomorphized?



this book is not for you.

[Note from marketing: this book is for anyone with a credit card.]

We know what you're thinking.

“How can this be a serious book?”

“What’s with all the graphics?”

“Can I actually learn it this way?”

And we know what your brain is thinking.

Your brain craves novelty. It’s always searching, scanning, *waiting* for something unusual. It was built that way, and it helps you stay alive.

Today, you’re less likely to be a tiger snack. But your brain’s still looking. You just never know.

So what does your brain do with all the routine, ordinary, normal things you encounter? Everything it *can* to stop them from interfering with the brain’s *real* job—recording things that *matter*. It doesn’t bother saving the boring things; they never make it past the “this is obviously not important” filter.

How does your brain *know* what’s important? Suppose you’re out for a day hike and a tiger jumps in front of you—what happens inside your head and body?

Neurons fire. Emotions crank up. *Chemicals surge*.

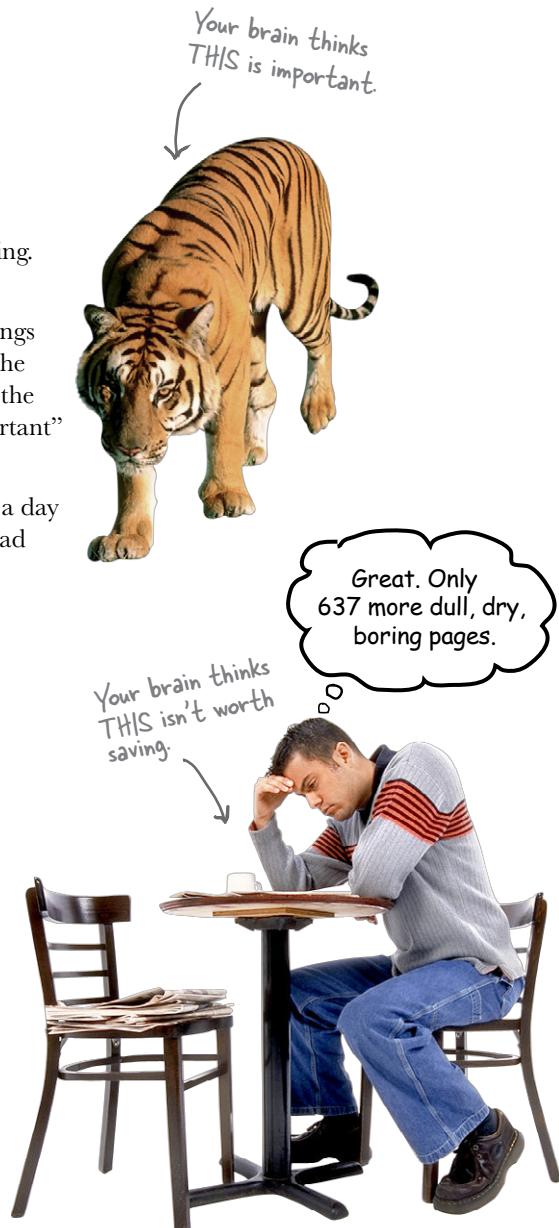
And that’s how your brain knows...

This must be important! Don’t forget it!

But imagine you’re at home, or in a library. It’s a safe, warm, tiger-free zone. You’re studying. Getting ready for an exam. Or trying to learn some tough technical topic your boss thinks will take a week, 10 days at the most.

Just one problem. Your brain’s trying to do you a big favor. It’s trying to make sure that this *obviously* non-important content doesn’t clutter up scarce resources. Resources that are better spent storing the really *big* things. Like tigers. Like the danger of fire. Like how you should never again snowboard in shorts.

And there’s no simple way to tell your brain, “Hey brain, thank you very much, but no matter how dull this book is, and how little I’m registering on the emotional Richter scale right now, I really *do* want you to keep this stuff around.”

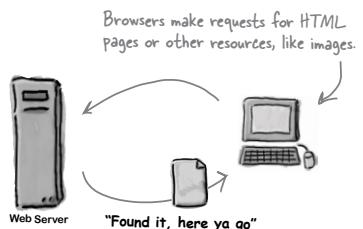


We think of a “Head First” reader as a learner.

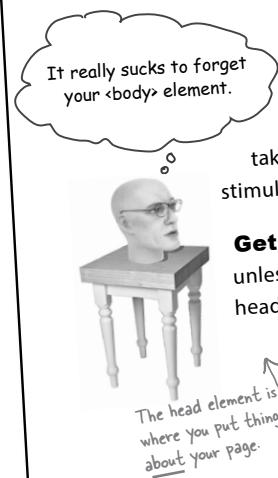
So what does it take to *learn* something? First, you have to *get it*, then make sure you *don’t forget it*. It’s not about pushing facts into your head. Based on the latest research in cognitive science, neurobiology, and educational psychology, *learning* takes a lot more than text on a page. We know what turns your brain on.

Some of the Head First learning principles:

Make it visual. Images are far more memorable than words alone, and make learning much more effective (up to 89% improvement in recall and transfer studies). It also makes things more understandable. **Put the words within or near the graphics** they relate to, rather than on the bottom or on another page, and learners will be up to twice as likely to solve problems related to the content.



Use a conversational and personalized style. In recent studies, students performed up to 40% better on post-learning tests if the content spoke directly to the reader, using a first-person, conversational style rather than taking a formal tone. Tell stories instead of lecturing. Use casual language. Don’t take yourself too seriously. Which would you pay more attention to: a stimulating dinner-party companion, or a lecture?



Get the learner to think more deeply. In other words, unless you actively flex your neurons, nothing much happens in your head. A reader has to be motivated, engaged, curious, and inspired to solve problems, draw conclusions, and generate new knowledge. And for that, you need challenges, exercises, and thought-provoking questions, and activities that involve both sides of the brain, and multiple senses.



Get—and keep—the reader’s attention. We’ve all had the “I really want to learn this, but I can’t stay awake past page one” experience. Your brain pays attention to things that are out of the ordinary, interesting, strange, eye-catching, unexpected. Learning a new, tough, technical topic doesn’t have to be boring. Your brain will learn much more quickly if it’s not.

Touch their emotions. We now know that your ability to remember something is largely dependent on its emotional content. You remember what you *care* about. You remember when you *feel* something. No, we’re not talking heart-wrenching stories about a boy and his dog. We’re talking emotions like surprise, curiosity, fun, “what the...?”, and the feeling of “I rule!” that comes when you solve a puzzle, learn something everybody else thinks is hard, or realize you know something that “I’m more technical than thou” Bob from engineering *doesn’t*.



Metacognition: thinking about thinking

If you really want to learn, and you want to learn more quickly and more deeply, pay attention to how you pay attention. Think about how you think. Learn how you learn.

Most of us did not take courses on metacognition or learning theory when we were growing up. We were *expected* to learn, but rarely *taught* how to learn.

But we assume that if you're holding this book, you really want to learn how to create web pages. And you probably don't want to spend a lot of time. And you want to *remember* what you read, and be able to apply it. And for that, you've got to *understand* it. To get the most from this book, or *any* book or learning experience, take responsibility for your brain. Your brain on *this* content.

The trick is to get your brain to see the new material you're learning as Really Important. Crucial to your well-being. As important as a tiger. Otherwise, you're in for a constant battle, with your brain doing its best to keep the new content from sticking.

So how **DO** you get your brain to think HTML & CSS are as important as a tiger?

There's the slow, tedious way, or the faster, more effective way. The slow way is about sheer repetition. You obviously know that you *are* able to learn and remember even the dullest of topics, if you keep pounding on the same thing. With enough repetition, your brain says, "This doesn't *feel* important to him, but he keeps looking at the same thing *over and over and over*, so I suppose it must be."

The faster way is to do **anything that increases brain activity**, especially different *types* of brain activity. The things on the previous page are a big part of the solution, and they're all things that have been proven to help your brain work in your favor. For example, studies show that putting words *within* the pictures they describe (as opposed to somewhere else in the page, like a caption or in the body text) causes your brain to try to make sense of how the words and picture relate, and this causes more neurons to fire. More neurons firing = more chances for your brain to *get* that this is something worth paying attention to, and possibly recording.

A conversational style helps because people tend to pay more attention when they perceive that they're in a conversation, since they're expected to follow along and hold up their end. The amazing thing is, your brain doesn't necessarily *care* that the "conversation" is between you and a book! On the other hand, if the writing style is formal and dry, your brain perceives it the same way you experience being lectured to while sitting in a roomful of passive attendees. No need to stay awake.

But pictures and conversational style are just the beginning.



Here's what WE did:

We used **pictures**, because your brain is tuned for visuals, not text. As far as your brain's concerned, a picture really *is* worth 1,024 words. And when text and pictures work together, we embedded the text *in* the pictures because your brain works more effectively when the text is *within* the thing the text refers to, as opposed to in a caption or buried in the text somewhere.

We used **redundancy**, saying the same thing in *different* ways and with different media types, and *multiple senses*, to increase the chance that the content gets coded into more than one area of your brain.

We used concepts and pictures in **unexpected** ways because your brain is tuned for novelty, and we used pictures and ideas with at least *some emotional content*, because your brain is tuned to pay attention to the biochemistry of emotions. That which causes you to *feel* something is more likely to be remembered, even if that feeling is nothing more than a little **humor**, **surprise**, or **interest**.

We used a personalized, **conversational style**, because your brain is tuned to pay more attention when it believes you're in a conversation than if it thinks you're passively listening to a presentation. Your brain does this even when you're *reading*.

We included more than 100 **activities**, because your brain is tuned to learn and remember more when you **do** things than when you *read* about things. And we made the exercises challenging-yet-doable, because that's what most *people* prefer.

We used **multiple learning styles**, because *you* might prefer step-by-step procedures, while someone else wants to understand the big picture first, while someone else just wants to see a code example. But regardless of your own learning preference, *everyone* benefits from seeing the same content represented in multiple ways.

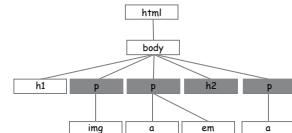
We include content for **both sides of your brain**, because the more of your brain you engage, the more likely you are to learn and remember, and the longer you can stay focused. Since working one side of the brain often means giving the other side a chance to rest, you can be more productive at learning for a longer period of time.

And we included **stories** and exercises that present **more than one point of view**, because your brain is tuned to learn more deeply when it's forced to make evaluations and judgments.

We included **challenges**, with exercises, and by asking **questions** that don't always have a straight answer, because your brain is tuned to learn and remember when it has to *work* at something. Think about it—you can't get your *body* in shape just by *watching* people at the gym. But we did our best to make sure that when you're working hard, it's on the *right* things. That **you're not spending one extra dendrite** processing a hard-to-understand example, or parsing difficult, jargon-laden, or overly terse text.

We used **people**. In stories, examples, pictures, etc., because, well, because *you're* a person. And your brain pays more attention to *people* than it does to *things*.

We used an **80/20** approach. We assume that if you're going to be a kick-butt web developer, this won't be your only book. So we don't talk about *everything*. Just the stuff you'll actually *need*.



Be the Browser

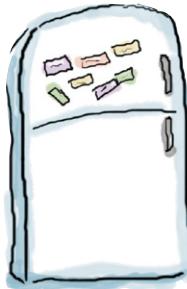


BULLET POINTS



Puzzles





Cut this out and stick it
on your refrigerator.

Here's what YOU can do to bend your brain into submission

So, we did our part. The rest is up to you. These tips are a starting point; listen to your brain and figure out what works for you and what doesn't. Try new things.

① Slow down. The more you understand, the less you have to memorize.

Don't just *read*. Stop and think. When the book asks you a question, don't just skip to the answer. Imagine that someone really *is* asking the question. The more deeply you force your brain to think, the better chance you have of learning and remembering.

② Do the exercises. Write your own notes.

We put them in, but if we did them for you, that would be like having someone else do your workouts for you. And don't just *look* at the exercises. **Use a pencil.** There's plenty of evidence that physical activity *while* learning can increase the learning.

③ Read the “There Are No Dumb Questions.”

That means all of them. They're not optional sidebars—**they're part of the core content!** Don't skip them.

④ Make this the last thing you read before bed. Or at least the last challenging thing.

Part of the learning (especially the transfer to long-term memory) happens *after* you put the book down. Your brain needs time on its own, to do more processing. If you put in something new during that processing time, some of what you just learned will be lost.

⑤ Drink water. Lots of it.

Your brain works best in a nice bath of fluid. Dehydration (which can happen before you ever feel thirsty) decreases cognitive function.

⑥ Talk about it. Out loud.

Speaking activates a different part of the brain. If you're trying to understand something, or increase your chance of remembering it later, say it out loud. Better still, try to explain it out loud to someone else. You'll learn more quickly, and you might uncover ideas you hadn't known were there when you were reading about it.

⑦ Listen to your brain.

Pay attention to whether your brain is getting overloaded. If you find yourself starting to skim the surface or forget what you just read, it's time for a break. Once you go past a certain point, you won't learn faster by trying to shove more in, and you might even hurt the process.

⑧ Feel something!

Your brain needs to know that this *matters*. Get involved with the stories. Make up your own captions for the photos. Groaning over a bad joke is *still* better than feeling nothing at all.

⑨ Create something!

Apply this to something new you're designing, or rework an older project. Just do *something* to get some experience beyond the exercises and activities in this book. All you need is a pencil and a problem to solve...a problem that might benefit from using HTML and CSS.

Read me

This is a learning experience, not a reference book. We deliberately stripped out everything that might get in the way of learning whatever it is we're working on at that point in the book. And the first time through, you need to begin at the beginning, because the book makes assumptions about what you've already seen and learned.

We begin by teaching basic HTML, then standards-based HTML5.

To write standards-based HTML, there are a lot of technical details you need to understand that aren't helpful when you're trying to learn the basics of HTML. Our approach is to have you learn the basic concepts of HTML first (without worrying about these details), and then, when you have a solid understanding of HTML, teach you to write standards-compliant HTML (the most recent version of which is HTML5). This has the added benefit that the technical details are more meaningful after you've already learned the basics.

It's also important that you be writing compliant HTML when you start using CSS, so we make a point of getting you to standards-based HTML before you begin any serious work with CSS.

We don't cover every single HTML element or attribute or CSS property ever created.

There are a *lot* of HTML elements, *a lot* of attributes, and *a lot* of CSS properties. Sure, they're all interesting, but our goal was to write a book that weighs less than the person reading it, so we don't cover them all here. Our focus is on the core HTML elements and CSS properties that *matter* to you, the beginner, and making sure that you really, truly, deeply understand how and when to use them. In any case, once you're done with *Head First HTML and CSS*, you'll be able to pick up any reference book and get up to speed quickly on all the elements and properties we left out.

This book advocates a clean separation between the structure of your pages and the presentation of your pages.

Today, serious web pages use HTML to structure their content, and CSS for style and presentation. Nineties-era pages often used a different model, one where HTML was used for both structure and style. This book teaches you to use HTML for structure and CSS for style; we see no reason to teach you outdated bad habits.

We encourage you to use more than one browser with this book.

While we teach you to write HTML and CSS that are based on standards, you'll still (and

probably always) encounter minor differences in the way web browsers display pages. So, we encourage you to pick at least two modern browsers and test your pages using them. This will give you experience in seeing the differences among browsers and in creating pages that work well in a variety of them.

We often use tag names for element names.

Rather than saying “the a element,” or “the ‘a’ element,” we use a tag name, like “the `<a>` element.” While this may not be technically correct (because `<a>` is an opening tag, not a full-blown element), it does make the text more readable, and we usually follow the name with the word “element” to avoid confusion.

The activities are NOT optional.

The exercises and activities are not add-ons; they’re part of the core content of the book. Some of them are to help with memory, some are for understanding, and some will help you apply what you’ve learned. **Don’t skip the exercises.** The crossword puzzles are the only things you don’t *have* to do, but they’re good for giving your brain a chance to think about the words in a different context.

The redundancy is intentional and important.

One distinct difference in a Head First book is that we want you to *really* get it. And we want you to finish the book remembering what you’ve learned. Most reference books don’t have retention and recall as a goal, but this book is about *learning*, so you’ll see some of the same concepts come up more than once.

The examples are as lean as possible.

Our readers tell us that it’s frustrating to wade through 200 lines of an example looking for the two lines they need to understand. Most examples in this book are shown within the smallest possible context, so that the part you’re trying to learn is clear and simple. Don’t expect all of the examples to be robust, or even complete—they are written specifically for learning, and aren’t always fully functional.

We’ve placed all the example files on the Web so you can download them. You’ll find them at <http://wickedlysmart.com/hfhtmlcss/>.

The Brain Power exercises don’t have answers.

For some of them, there is no right answer, and for others, part of the learning experience of the Brain Power activities is for you to decide if and when your answers are right. In some of the Brain Power exercises, you will find hints to point you in the right direction.

Tech reviewers (first edition)

Louise Barr



Joe Konior



Valentin Crettaz



Corey McGlone



Barney Marispini



Pauline McNamara



Eiffel Tower



Johannes de Jong

Pauline gets the "kick-ass reviewer" award.

Fearless leader
of the Extreme
Review Team

Marcus Green



Ike Van Atta



David O'Meara



Our reviewers:

We're extremely grateful for our technical review team. **Johannes de Jong** organized and led the whole effort, acted as "series dad," and made it all work smoothly. **Pauline McNamara**, "co-manager" of the effort, held things together and was the first to point out when our examples were a little more "baby boomer" than hip. The whole team proved how much we needed their technical expertise and attention to detail. **Valentin Crettaz, Barney Marispini, Marcus Green, Ike Van Atta, David O'Meara, Joe Konior, and Corey McGlone** left no stone unturned in their review and the book is much better for it. You guys rock! And further thanks to **Corey** and **Pauline** for never letting us slide on our often too formal (or we should just say it, incorrect) punctuation. A shout-out to JavaRanch as well for hosting the whole thing.

A big thanks to **Louise Barr**, our token web designer, who kept us honest on our designs and on our use of HTML and CSS (although you'll have to blame us for the actual designs).

Acknowledgments (first edition)*

Even more technical review:

We're also extremely grateful to our esteemed technical reviewer **David Powers**. We have a real love/hate relationship with David because he made us work so hard, but the result was *oh so worth it*. The truth be told, based on David's comments, we made significant changes to this book and it is technically twice the book it was before. Thank you, David.

At O'Reilly:

Our biggest thanks to our editor, **Brett McLaughlin**, who cleared the path for this book, removed every obstacle to its completion, and sacrificed family time to get it done. Brett also did hard editing time on this book (not an easy task for a Head First title). Thanks, Brett; this book wouldn't have happened without you.



Brett McLaughlin

Our sincerest thanks to the whole O'Reilly team: **Greg Corrin**, **Glenn Bisignani**, **Tony Artuso**, and **Kyle Hart** all led the way on marketing and we appreciate their out-of-the-box approach. Thanks to **Ellie Volkhausen** for her inspired cover design that continues to serve us well, and to **Karen Montgomery** for stepping in and bringing life to this book's cover. Thank you, as always, to **Colleen Gorman** for her hardcore copyedit (and for keeping it all fun). And we couldn't have pulled off a color book like this without **Sue Willing** and **Claire Cloutier**.

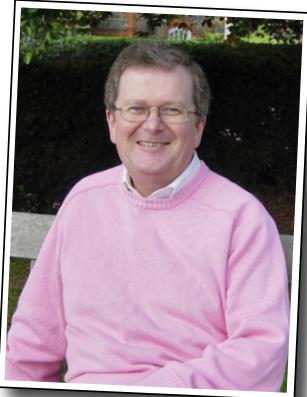
No Head First acknowledgment would be complete without thanking **Mike Loukides** for shaping the Head First concept into a series, and to **Tim O'Reilly** for always being there and his continued support. Finally, thanks to **Mike Hendrickson** for bringing us into the Head First family and having the faith to let us run with it.

Kathy Sierra and Bert Bates:

Last, and anything but least, to **Kathy Sierra** and **Bert Bates**, our partners in crime and the BRAINS who created the series. Thanks, guys, for trusting us *even more* with your baby. We hope once again we've done it justice. The three-day jam session was the highlight of writing the book, we hope to repeat it soon. Oh, and next time around, can you give LTJ a call and tell him he's just going to have to make a trip back to Seattle?

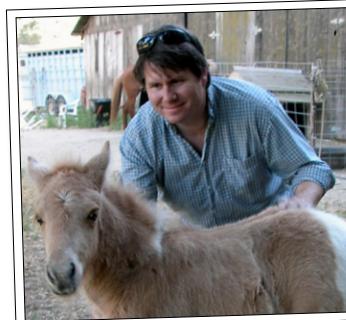
*The large number of acknowledgments is because we're testing the theory that everyone mentioned in a book acknowledgment will buy at least one copy, probably more, what with relatives and everything. If you'd like to be in the acknowledgment of our *next* book, and you have a large family, write to us.

↓ Esteemed Reviewer
David Powers



Don't let the sweater fool you—this guy is hardcore (technically of course).

Bert Bates



Kathy Sierra



Hard at work researching
Head First Parelli

↑ Kara

Tech reviewers (second edition)

We couldn't sleep at night without knowing that our high-powered HTML & CSS reviewer, **David Powers**, has scoured this book for inaccuracies. Truth is, so many years had passed since the first edition that we had to hire a private detective to locate him (it's a long story, but he was finally located in his underground HTML & CSS lair and research lab). Anyway, more seriously, while all the technical faults in this book sit solely with the authors (that's us), we can assure you in every case David tried to make sure we did things right. Once again, David was instrumental in the writing of this book.

We're extremely grateful for everyone on our technical review team. **Joe Konior** joined us once again for this edition, along with **Dawn Griffiths** (co-author of *Head First C*), and **Shelley Powers** (an HTML & CSS "power"house who has been writing about the Web for years). Once again, you all rock! Your feedback was amazingly thorough, detailed, and helpful. Thank you.



Dawn Griffiths



Joe Konior



Mike Hendrickson



Lou Barr



Acknowledgments (second edition)

Our biggest thanks to our chief editor, **Mike Hendrickson**, who made this book happen in every way (other than actually writing it), was there for us the entire journey, and more importantly (the biggest thing any editor can do) totally trusted us to get it done! Thanks, Mike; none of our books would have happened without you. You've been our champion for well over a decade and we love you for it!

Of course it takes a village to publish a book, and behind the scenes a talented and friendly group at O'Reilly made it all happen. Our sincerest thanks to the whole O'Reilly team: **Kristen Borg** (production editor extraordinaire); the brilliant **Rachel Monaghan** (proofreader); **Ron Strauss** for his meticulous index; **Rebecca Demarest** for illustration help; **Karen Montgomery**, ace cover designer; and last but definitely not least, **Louise Barr**, who always helps our pages look better.

Safari® Books Online



Safari® Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at <http://my.safaribooksonline.com>.

1 getting to know HTML

The Language of the Web



The only thing that is standing between you and getting yourself on the Web is learning to speak the lingo:

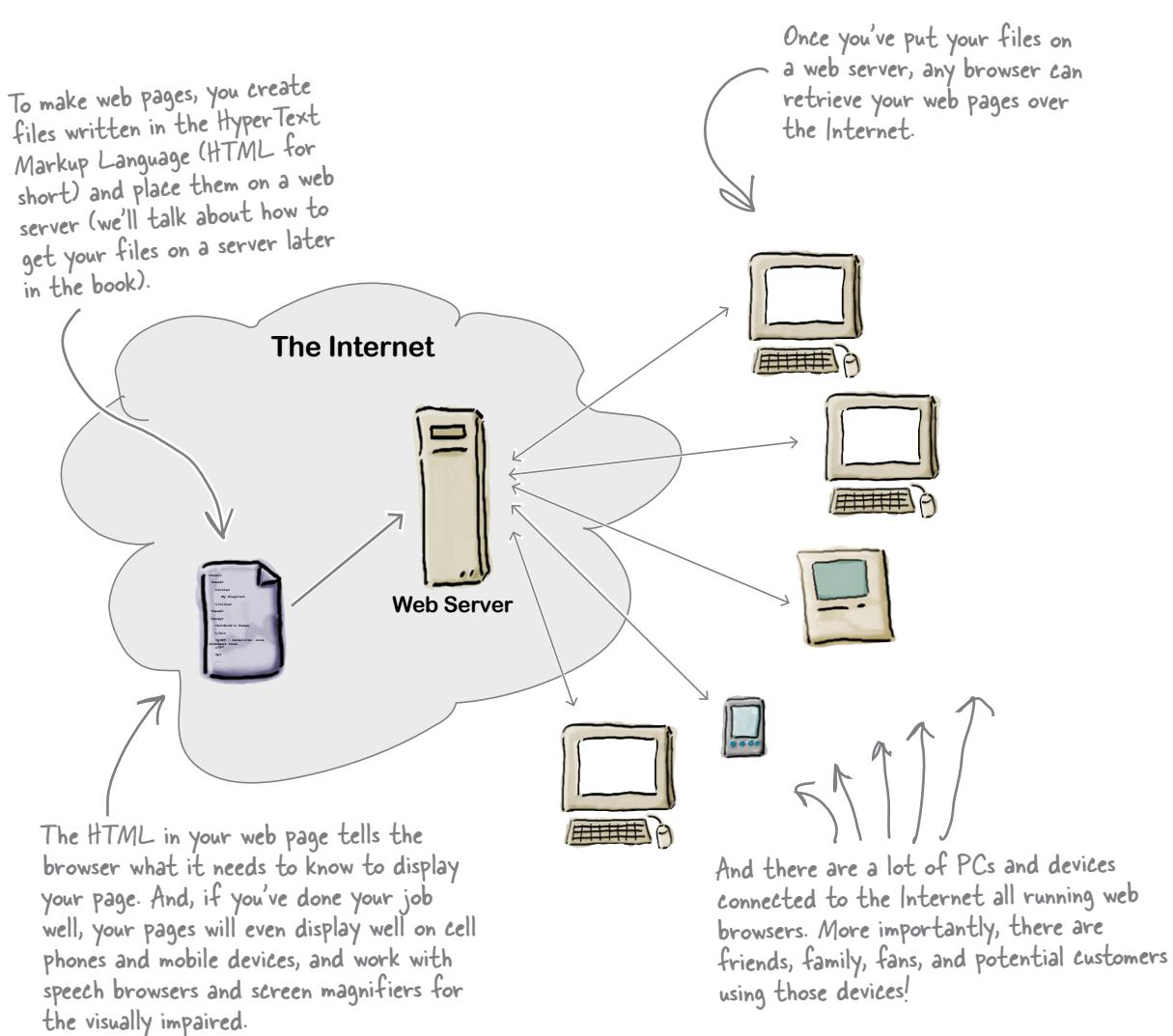
HyperText Markup Language, or HTML for short. So, get ready for some language lessons. After this chapter, not only are you going to understand some basic **elements** of HTML, but you'll also be able to speak HTML with a little **style**. Heck, by the end of this book you'll be talking HTML like you grew up in Webville.

The Web

~~Video killed the radio star~~

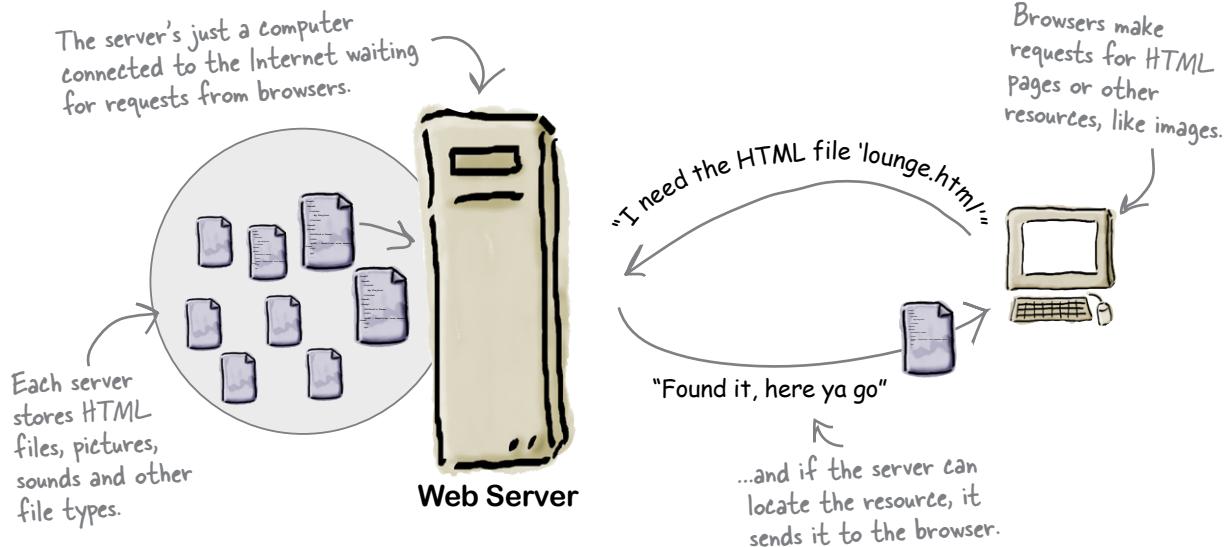
Want to get an idea out there? Sell something? Just need a creative outlet? Turn to the Web—we don't need to tell you it has become the universal form of communication. Even better, it's a form of communication **YOU** can participate in.

But if you really want to use the Web effectively, you've got to know a few things about **HTML**—not to mention, a few things about how the Web works too. Let's take a look from 30,000 feet:



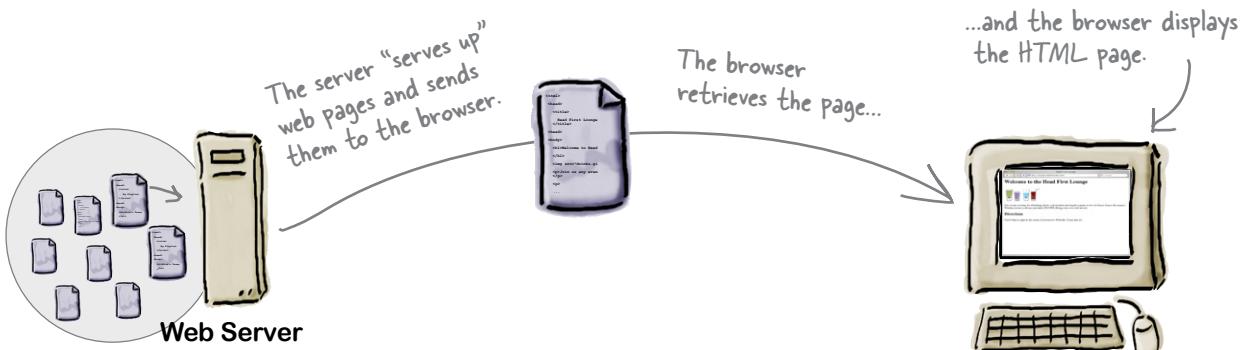
What does the web server do?

Web servers have a full-time job on the Internet, tirelessly waiting for requests from web browsers. What kinds of requests? Requests for web pages, images, sounds, or maybe even a video. When a server gets a request for any of these resources, the server finds the resource, and then sends it back to the browser.



What does the web browser do?

You already know how a browser works: you're surfing around the Web and you click on a link to visit a page. That click causes your browser to request an HTML page from a web server, retrieve it, and display the page in your browser window.



But how does the browser know how to display a page? That's where HTML comes in. HTML tells the browser all about the content and structure of the page. Let's see how that works...

What you write (the HTML)

So, you know HTML is the key to getting a browser to display your pages, but what exactly does HTML look like? And what does it do?

Let's have a look at a little HTML...imagine you're going to create a web page to advertise the *Head First Lounge*, a local hangout with some good tunes, refreshing elixirs, and wireless access. Here's what you'd write in HTML:

```
<html>
  <head>
    <title>Head First Lounge</title> A
  </head>
  <body>
    <h1>Welcome to the Head First Lounge</h1> B
     C
    <p>
      D Join us any evening for refreshing elixirs,
      conversation and maybe a game or
      two of <em>Dance Dance Revolution</em>. E
      Wireless access is always provided;
      BYOWS (Bring your own web server).
    </p>
    <h2>Directions</h2> F
    <p>
      G You'll find us right in the center of
      downtown Webville. Come join us!
    </p>
  </body>
</html>
```



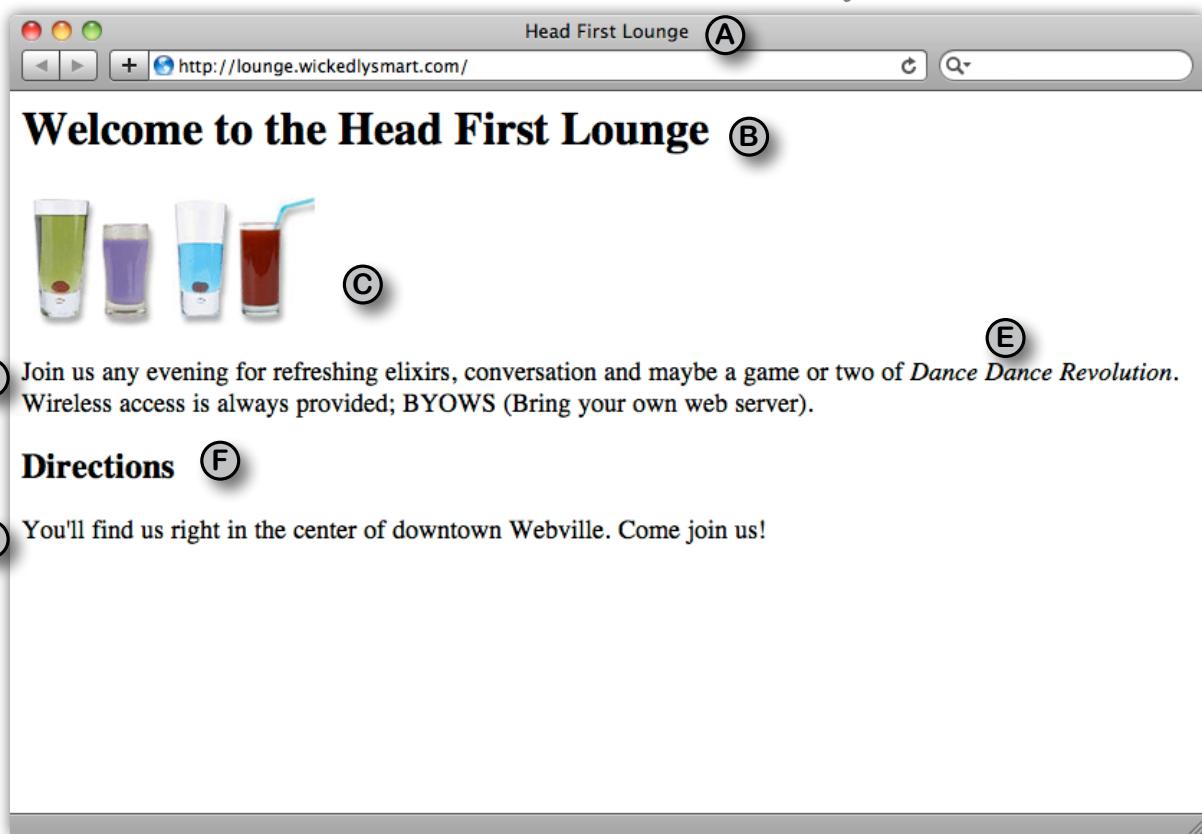
We don't expect you to know HTML yet.

At this point you should just be getting a feel for what HTML looks like; we're going to cover everything in detail in a bit. For now, study the HTML and see how it gets represented in the browser on the next page. Be sure to pay careful attention to each letter annotation and how and where it is displayed in the browser.

What the browser creates

When the browser reads your HTML, it interprets all the *tags* that surround your text. Tags are just words or characters in angle brackets, like `<head>`, `<p>`, `<h1>`, and so on. The tags tell the browser about the *structure and meaning* of your text. So rather than just giving the browser a bunch of text, with HTML you can use tags to tell the browser what text is in a heading, what text is a paragraph, what text needs to be emphasized, or even where images need to be placed.

Let's check out how the browser interprets the tags in the Head First Lounge:



there are no Dumb Questions

Q: So HTML is just a bunch of tags that I put around my text?

A: For starters. Remember that HTML stands for HyperText Markup Language, so HTML gives you a way to “mark up” your text with tags that tell the browser how your text is structured. But there is also the HyperText aspect of HTML, which we’ll talk about a little later in the book.

Q: How does the browser decide how to display the HTML?

A: HTML tells your browser about the structure of your document: where the headings are, where the paragraphs are, what text needs emphasis, and so on. Given this information, browsers have built-in default rules for how to display each of these elements.

But you don’t have to settle for the default settings. You can add your own style and formatting rules with CSS that determine font, colors, size, and a lot of other characteristics of your page. We’ll get back to CSS later in the chapter.

Q: The HTML for the Head First Lounge has all kinds of indentation and spacing, and yet I don’t see that when it is displayed in the browser. How come?

A: Correct, and good catch. Browsers ignore tabs, returns, and most spaces in HTML documents. Instead, they rely on your markup to determine where line and paragraph breaks occur.

So why did we insert our own formatting if the browser is just going to ignore it? To help us more easily read the document when we’re editing the HTML. As your

HTML documents become more complicated, you’ll find a few spaces, returns, and tabs here and there really help to improve the readability of the HTML.

Q: So there are two levels of headings, <h1> and a subheading <h2>?

A: Actually there are six, <h1> through <h6>, which the browser typically displays in successively smaller font sizes. Unless you are creating a complex and large document, you typically won’t use headings beyond <h3>.

Q: Why do I need the <html> tag? Isn’t it obvious this is an HTML document?

A: The <html> tag tells the browser your document is actually HTML. While some browsers will forgive you if you omit it, some won’t, and as we move toward “industrial-strength HTML” later in the book, you’ll see it is quite important to include this tag.

Q: What makes a file an HTML file?

A: An HTML file is a simple text file. Unlike a word processing file, there is no special formatting embedded in it. By convention, we add an “.html” to the end of the filename to give the operating system a better idea of what the file is. But, as you’ve seen, what really matters is what we put inside the file.

Q: Everyone is talking about HTML5. Are we using it? If so, why aren’t we saying “HTML-FIVE” instead of “HTML”?

A: You’re learning about HTML, and HTML5 just happens to be the latest version of HTML. HTML5 has had a lot of attention recently, and that’s because it simplifies

many of the ways we write HTML and enables some new functionality, which we’re going to cover in this book. It also provides some advanced features through its JavaScript application programming interfaces (APIs), and those are covered in Head First HTML5 Programming.

Q: Markup seems silly. What-you-see-is-what-you-get applications have been around since, what, the ’70s? Why isn’t the Web based on a format like Microsoft Word or a similar application?

A: The Web is created out of text files without any special formatting characters. This enables any browser in any part of the world to retrieve a web page and understand its contents. There are WYSIWYG applications out there like Dreamweaver, and they work great. But in this book we’re going to take it down to the bare metal, and start with text. Then you’re in good shape to understand what your Dreamweaver application is doing behind the scenes.

Q: Is there any way to put comments to myself in HTML?

A: Yes, if you place your comments in between <!-- and --> the browser will totally ignore them. Say you wanted to write a comment “Here’s the beginning of the lounge content.” You’d do that like this:

```
<!-- Here's the beginning of  
the lounge content -->
```

Notice that you can put comments on multiple lines. Keep in mind anything you put between the “<!--” and the “-->”, even HTML, will be ignored by the browser.



Sharpen your pencil

You're closer to learning HTML than you think...

Here's the HTML for the Head First Lounge again. Take a look at the tags and see if you can guess what they tell the browser about the content. Write your answers in the space on the right; we've already done the first couple for you.

<html>	Tells the browser this is the start of HTML.
<head>	Starts the page "head" (more about this later)
<title>Head First Lounge</title>	
</head>	
<body>	
<h1>Welcome to the Head First Lounge</h1>	
<p>	
Join us any evening for refreshing elixirs, conversation and maybe a game or two of Dance Dance Revolution. Wireless access is always provided; BYOWS (Bring your own web server).	
</p>	
<h2>Directions</h2>	
<p>	
You'll find us right in the center of downtown Webville. Come join us!	
</p>	
</body>	
</html>	



Sharpen your pencil

Solution

```
<html>
  <head>
    <title>Head First Lounge</title>
  </head>
  <body>
    <h1>Welcome to the Head First Lounge</h1>
    
    <p>
      Join us any evening for refreshing elixirs,
      conversation and maybe a game or
      two of <em>Dance Dance Revolution</em>.
      Wireless access is always provided;
      BYOWS (Bring your own web server).
    </p>
    <h2>Directions</h2>
    <p>
      You'll find us right in the center of
      downtown Webville. Come join us!
    </p>
  </body>
</html>
```

Tells the browser this is the start of HTML.

Starts the page "head".

Gives the page a title.

End of the head.

Start of the body of page.

Tells browser that "Welcome to..." is a heading.

Places the image "drinks.gif" here.

Start of a paragraph.

Puts emphasis on Dance Dance Revolution.

End of paragraph.

Tells the browser that "Directions" is a subheading.

Start of another paragraph.

End of paragraph.

End of the body.

Tells the browser this is the end of the HTML.



Your big break at Starbuzz Coffee

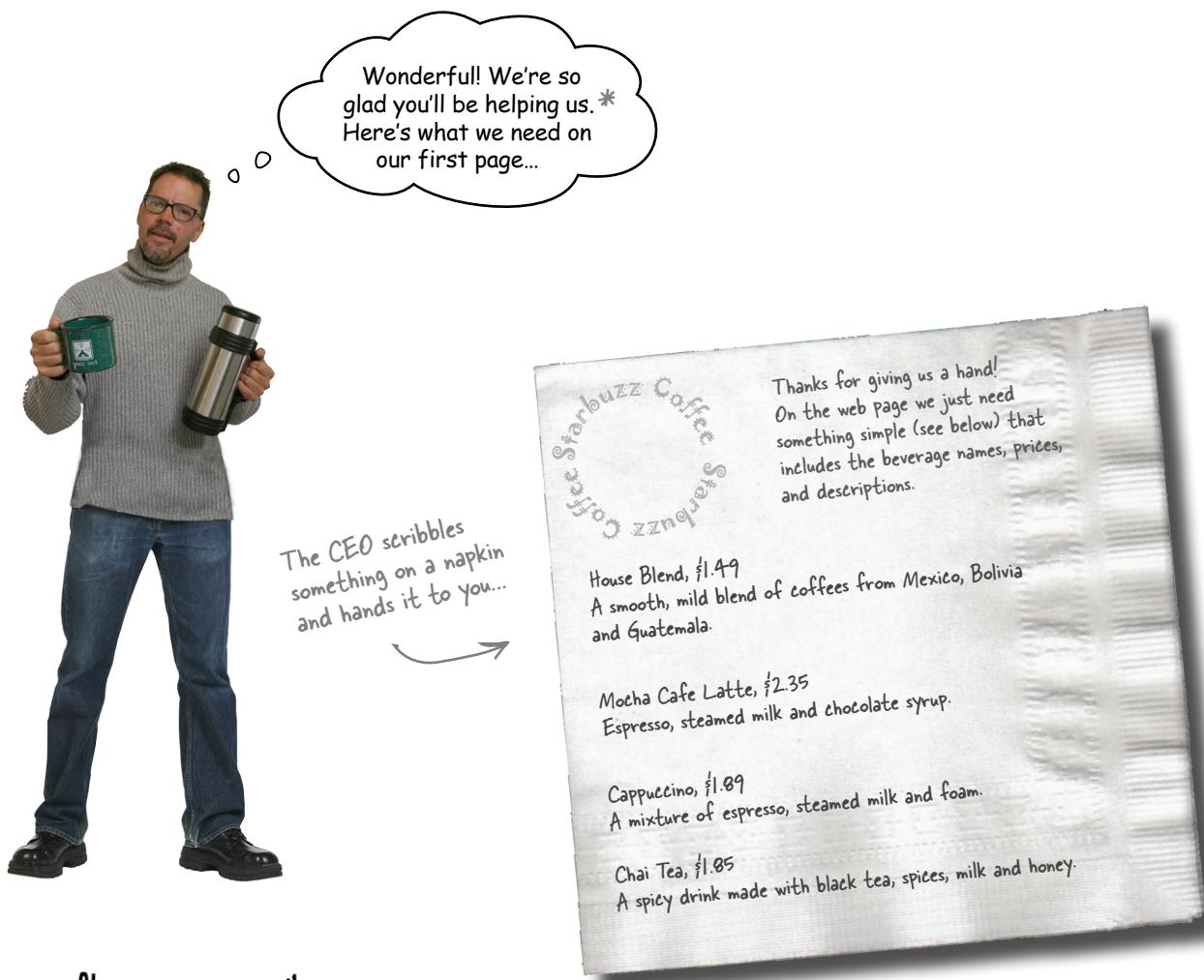
Starbuzz Coffee has made a name for itself as the fastest growing coffee shop around. If you've seen one on your local corner, look across the street—you'll see another one.

In fact, they've grown so quickly, they haven't even managed to put up a web page yet...and therein lies your big break: By chance, while buying your Starbuzz Chai Tea, you run into the Starbuzz CEO...



Decisions, decisions.
Check your first priority below (choose only one):

- A. Give dog a bath.
- B. Finally get my checking account balanced.
- C. Take the Starbuzz gig and launch BIG-TIME web career.
- D. Schedule dentist appointment.



Sharpen your pencil

Take a look at the napkin. Can you determine the *structure* of it? In other words, are there obvious headings? Paragraphs? Is it missing anything like a title?

Go ahead and mark up the napkin (using your pencil) with any structure you see, and add anything that is missing.

You'll find our answers at the end of Chapter 1.

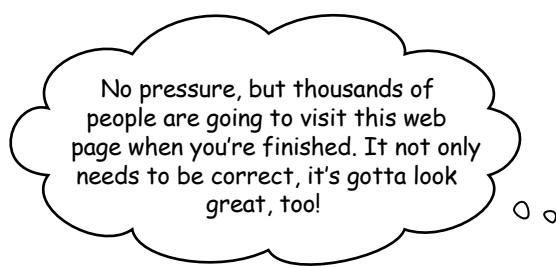
* If by chance you chose option A, B, or D on the previous page, we recommend you donate this book to a good library, use it as kindling this winter, or what the heck, go ahead and sell it on Amazon and make some cash.

Creating the Starbuzz web page

Of course, the only problem with all this is that you haven't actually created any web pages yet. But that's why you decided to dive head first into HTML, right?

No worries, here's what you're going to do on the next few pages:

- ① Create an HTML file using your favorite text editor.**
- ② Type in the menu the Starbuzz CEO wrote on the napkin.**
- ③ Save the file as “index.html”.**
- ④ Open the file “index.html” in your favorite browser, step back, and watch the magic happen.**



Creating an HTML file (Mac)

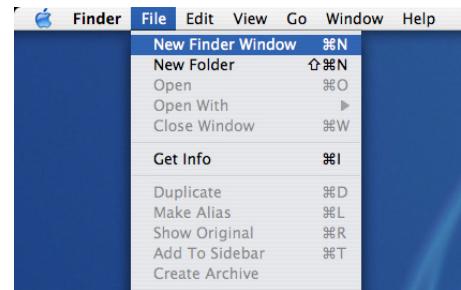
All HTML files are text files. To create a text file, you need an application that allows you to create plain text without throwing in a lot of fancy formatting and special characters. You just need plain, pure text.

We'll use TextEdit on the Mac in this book; however, if you prefer another text editor, that should work fine as well. And, if you're running Windows, you'll want to skip ahead a couple of pages to the Windows instructions.

Step one:

Navigate to your **Applications** folder

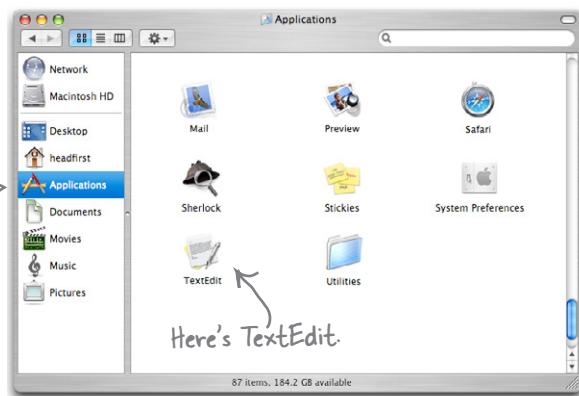
The TextEdit application is in the *Applications* folder. The easiest way to get there is to choose New Finder Window from the Finder's File menu and then look for the Application directly in your shortcuts. When you've found it, click on Applications.



Step two:

Locate and run **TextEdit**

You'll probably have lots of applications listed in your *Applications* folder, so scroll down until you see TextEdit. To run the application, double-click on the TextEdit icon.



Step three (optional):

Keep **TextEdit** in your Dock

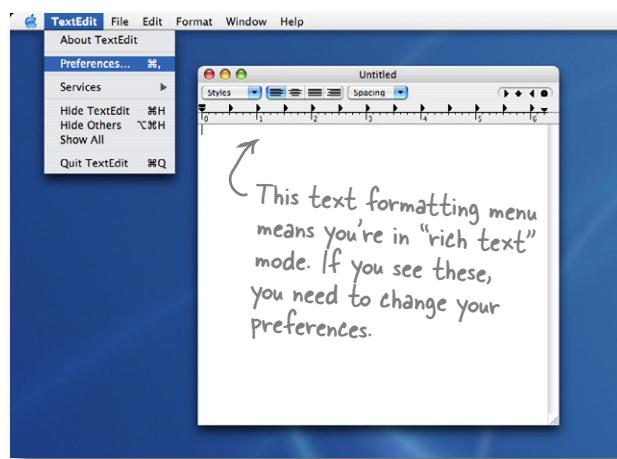
If you want to make your life easier, click and hold on the TextEdit icon in the Dock (this icon appears once the application is running). When it displays a pop-up menu, choose Options, then "Keep in Dock." That way, the TextEdit icon will always appear in your Dock and you won't have to hunt it down in the *Applications* folder every time you need to use it.



Step four:

Change your TextEdit Preferences

By default, TextEdit is in “rich text” mode, which means it will add its own formatting and special characters to your file when you save it—not what you want. So, you’ll need to change your TextEdit Preferences so that TextEdit saves your work as a pure text file. To do this, first choose the Preferences menu item from the TextEdit menu.



Step five:

Set Preferences for Plain text

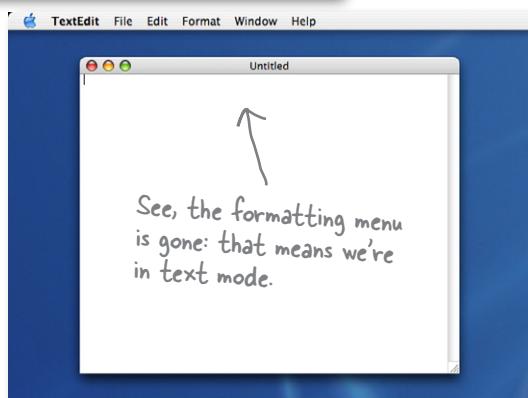
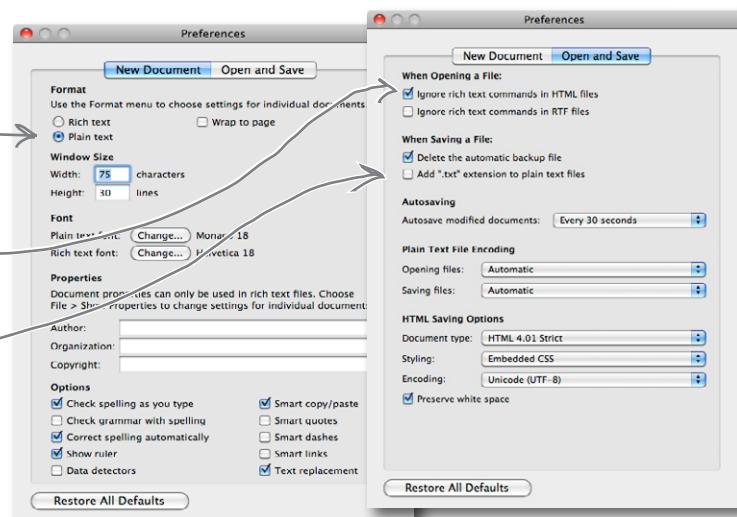
Once you see the Preferences dialog box, there are three things you need to do.

First, choose “Plain text” as the default editor mode in the New Document tab.

In the “Open and Save” tab, make sure “Ignore rich text commands in HTML files” is checked.

Last, make sure that the “Add .txt extension to plain text files” is **unchecked**.

That's it; to close the dialog box, click on the red button in the top-left corner.



Step six:

Quit and restart

Now quit out of TextEdit by choosing Quit from the TextEdit menu, and then restart the application. This time, you’ll see a window with no fancy text formatting menus at the top. You’re now ready to create some HTML.

Creating an HTML file (Windows)

If you're reading this page you must be a Windows 7 user. If you're not, you might want to skip a couple of pages ahead. Or, if you just want to sit in the back and not ask questions, we're okay with that too.

To create HTML files in Windows 7, we're going to use Notepad—it ships with every copy of Windows, the price is right, and it's easy to use. If you've got your own favorite editor that runs on Windows 7, that's fine too; just make sure you can create a plain-text file with an ".html" extension.

Assuming you're using Notepad, here's how you're going to create your first HTML file.

Step one:

Open the **Start** menu and navigate to Notepad.

You'll find the Notepad application in *Accessories*. The easiest way to get there is to click on the Start menu, then on All Programs, then Accessories. You'll see Notepad listed there.



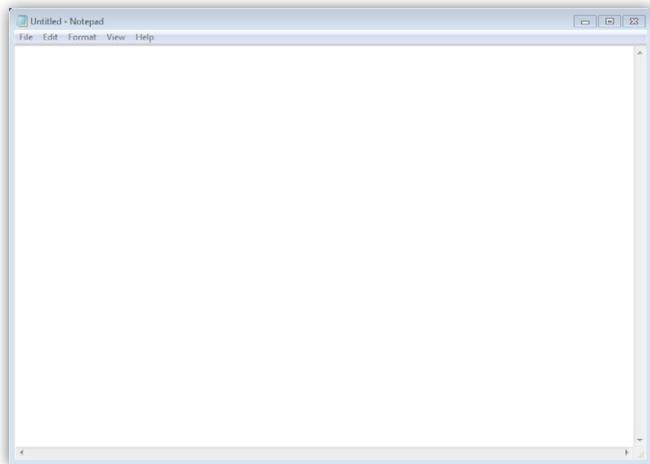
Or another version of Windows

If you're using another
version of Windows, you'll
find Notepad there as well.

Step two:

Open Notepad.

Once you've located Notepad in the *Accessories* folder, go ahead and click on it. You'll see a blank window ready for you to start typing HTML.



But recommended



Step three (optional):

Don't hide extensions of well-known file types.

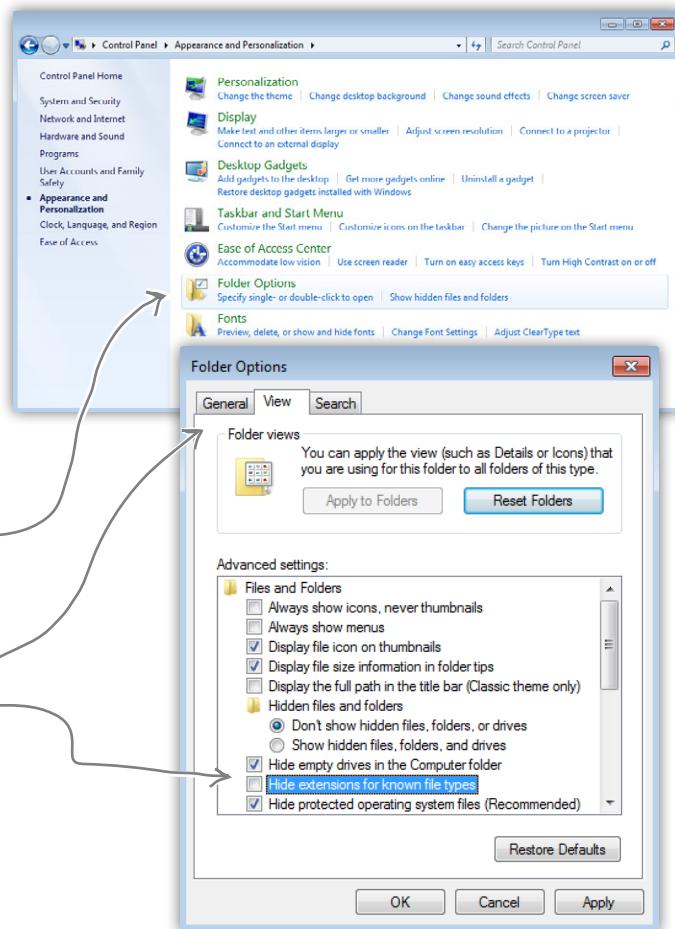
By default, Windows File Explorer hides the file extensions of well-known file types. For example, a file named "Irule.html" will be shown in the Explorer as "Irule" without its ".html" extension.

It's much less confusing if Windows shows you these extensions, so let's change your folder options so you can see them.

First, open Folder Options by clicking the Start button, clicking Control Panel, clicking "Appearance and Personalization," and then clicking Folder Options.

Next, in the View tab, under "Advanced settings," scroll down until you see "Hide extensions for known file types" and *uncheck* this option.

That's it. Click on the OK button to save the preference and you'll now see the file extensions in the Explorer.



^{there are no} Dumb Questions

Q: Why am I using a simple text editor?
Aren't there powerful tools like Dreamweaver
and Expression Web for creating web pages?

A: You're reading this book because you want to understand the true technologies used for web pages, right? Now those are all great tools, but they do a lot of the work for you, and until you are a master of HTML and CSS, you want to learn this stuff without a big tool getting in your way.

Once you're a master, however, these tools do provide some nice features like syntax checking and previews. At that point, when you view the "code" window, you'll understand everything in it, and you'll find that changes to the raw HTML and CSS are often a lot faster than going through a user interface. You'll also find that as standards change, these tools aren't always updated right away and may not support the most recent standards until their next release cycle. Since you'll know how to change the HTML and CSS without the tool, you'll be able to keep up with the latest and greatest all the time.

There are many more fully featured editors that include great features like clips (for automatically inserting bits of HTML you write often), preview (for previewing directly in the editor before you test in the browser), syntax coloring (so tags are a different color from content), and much more. Once you get the hang of writing basic HTML and CSS in a simple editor, it may be worth checking out one of the fancier editors, such as Coda, TextMate, CoffeeCup, or Aptana Studio. There are many out there to choose from (both free and not).

Q: I get the editor, but what browser am I supposed to be using? There are so many—Internet Explorer, Chrome, Firefox, Opera, Safari—what's the deal?

A: The simple answer: use whatever browser you like. HTML and CSS are industry standards, which means that all browsers try to support HTML and CSS in the same way (just make sure you are using the newest version of the browser for the best support).

The complex answer: in reality there are slight differences in the way browsers handle your pages. If you've got users who will be accessing your pages in a variety of browsers, then always test your web page in several different browsers. Some pages will look exactly the same; some won't. The more advanced you become with HTML and CSS, the more these slight differences may matter to you, and we'll get into some of these subtleties throughout the book.

Any of the major browsers—Internet Explorer, Chrome, Firefox, Opera, and Safari—will work for most examples (except where noted); they are all modern browsers with great HTML and CSS support. And as a web developer, you'll be expected to test your code in more than one browser, so we encourage you to download and get familiar with at least two!

Q: I'm creating these files on my own computer—how am I going to view these on the Web?

A: That's one great thing about HTML: you can create files and test them on your own computer and then later publish them on the Web. Right now, we're going to worry about how to create the files and what goes in them. We'll come back to getting them on the Web a bit later.



Meanwhile, back at Starbuzz Coffee...

Okay, now that you know the basics of creating a plain-text file, you just need to get some content into your text editor, save it, and then load it into your browser.

Start by typing in the beverages straight from the CEO's napkin; these beverages are the content for your page. You'll be adding some HTML markup to give the content some structure in a bit, but for now, just get the basic content typed in. While you're at it, go ahead and add "Starbuzz Coffee Beverages" at the top of the file.

Type in the info from
the napkin like this.



Untitled - Notepad

```
File Edit Format View Help
Starbuzz Coffee Beverages
House Blend, $1.49
A smooth, mild blend of coffees from Mexico, Bolivia and Guatemala.

Mocha Cafe Latte, $2.35
Espresso, steamed milk and chocolate syrup.

Cappuccino, $1.89
A mixture of espresso, steamed milk and foam.

Chai Tea, $1.85
A spicy drink made with black tea, spices, milk and honey.|
```

Untitled.txt

Starbuzz Coffee Beverages

```
House Blend, $1.49
A smooth, mild blend of coffees from Mexico, Bolivia and Guatemala.

Mocha Cafe Latte, $2.35
Espresso, steamed milk and chocolate syrup.

Cappuccino, $1.89
A mixture of espresso, steamed milk and foam.

Chai Tea, $1.85
A spicy drink made with black tea, spices, milk and honey.
```

Mac

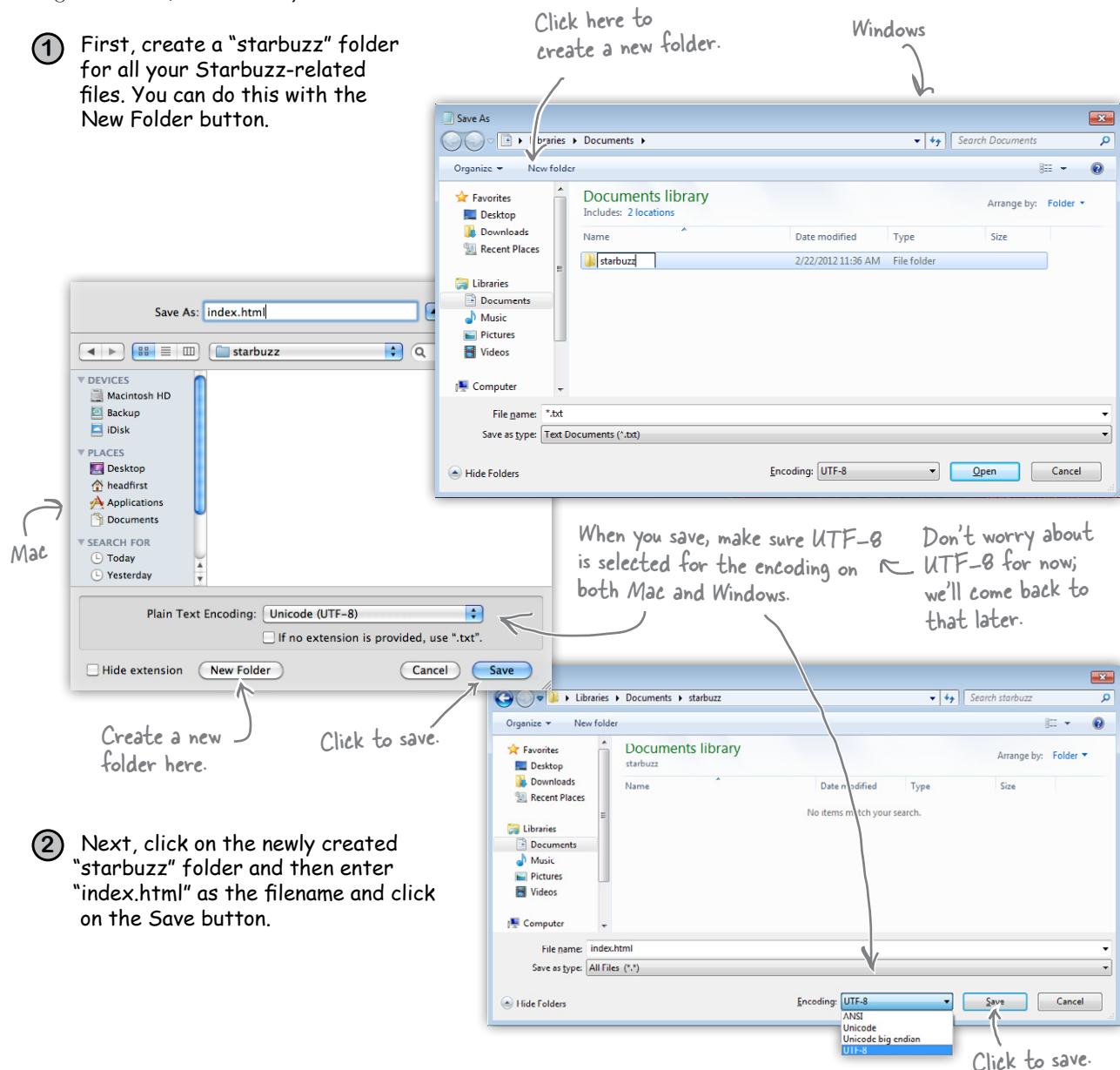
Windows

Saving your work

Once you've typed in the beverages from the CEO's napkin, you're going to save your work in a file called "index.html". Before you do that, you'll want to create a folder named "starbuzz" to hold the site's files.

To get this all started, choose Save from the File menu and you'll see a Save As dialog box. Then, here's what you need to do:

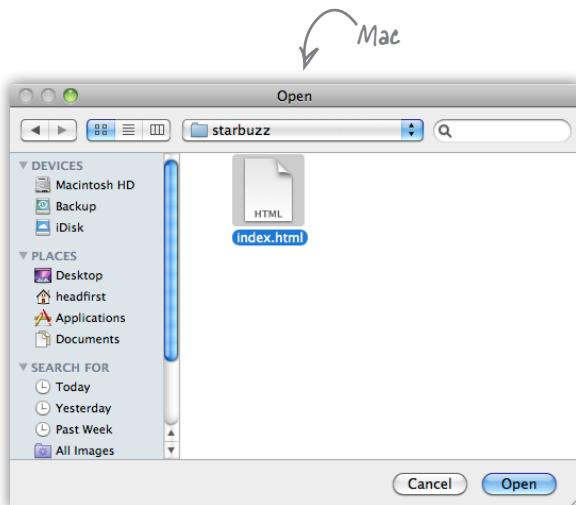
- ① First, create a "starbuzz" folder for all your Starbuzz-related files. You can do this with the New Folder button.



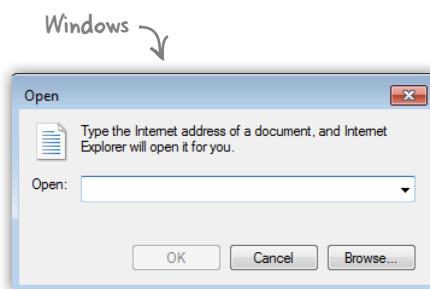
- ② Next, click on the newly created "starbuzz" folder and then enter "index.html" as the filename and click on the Save button.

Opening your web page in a browser

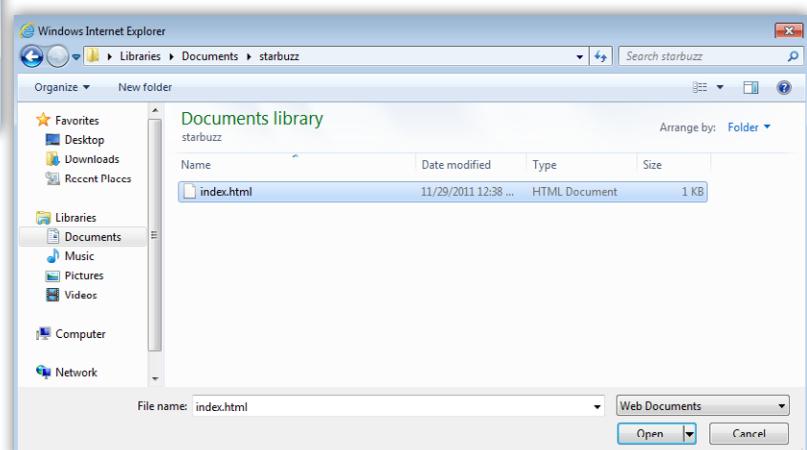
Are you ready to open your first web page? Using your favorite browser, choose “Open File...” (or “Open...” using Windows 7 and Internet Explorer) from the File menu and navigate to your “index.html” file. Select it and click Open.



On the Mac, navigate to your file, and select it by clicking on the file icon and then on the Open button.



In Windows Internet Explorer it's a two-step process. First, you'll get the Open dialog box.

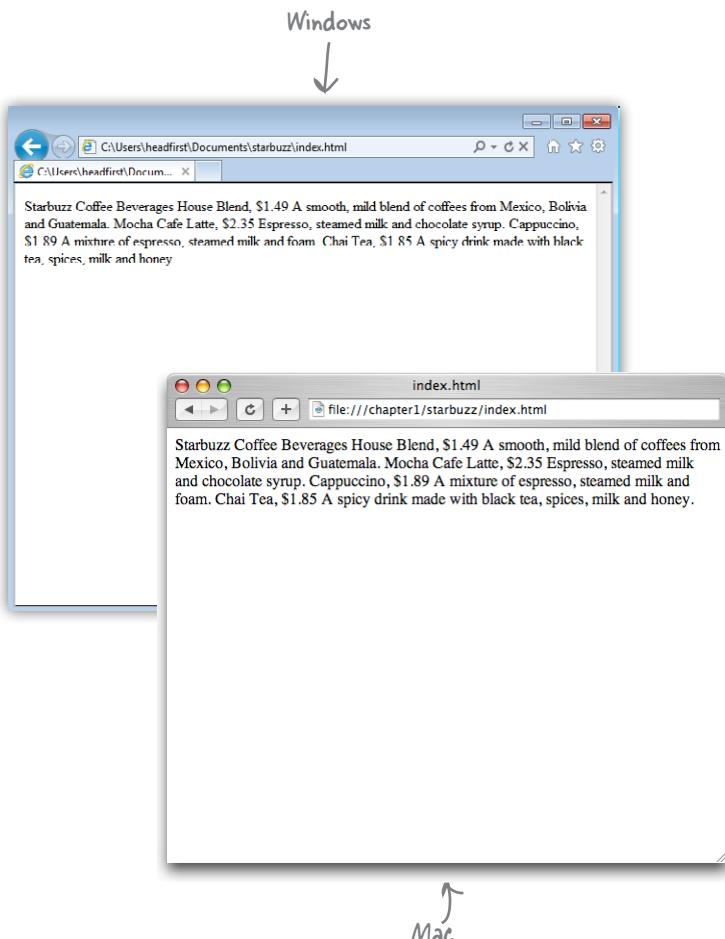


Then click Browse to get a browse dialog and navigate to where you saved your file.

Take your page for a test drive



Success! You've got the page loaded in the browser, although the results are a little...uh...unsatisfying. But that's just because all you've done so far is go through the mechanics of creating a page and viewing it in the browser. And so far, you've only typed in the *content* of the web page. That's where HTML comes in. HTML gives you a way to tell the browser about the *structure* of your page. What's structure? As you've already seen, it is a way of marking up your text so that the browser knows what's a heading, what text is in a paragraph, what text is a subheading, and so on. Once the browser knows a little about the structure, it can display your page in a more meaningful and readable manner.



Mac



Depending on your operating system and browser, often you can just double-click the HTML file or drag it on top of the browser icon to open it. Much simpler.



Markup Magnets

So, let's add that structure...

Your job is to add structure to the text from the Starbuzz napkin. Use the fridge magnets at the bottom of the page to mark up the text so that you've indicated which parts are headings, subheadings and paragraph text. We've already done a few to get you started. You won't need all the magnets below to complete the job; some will be left over.

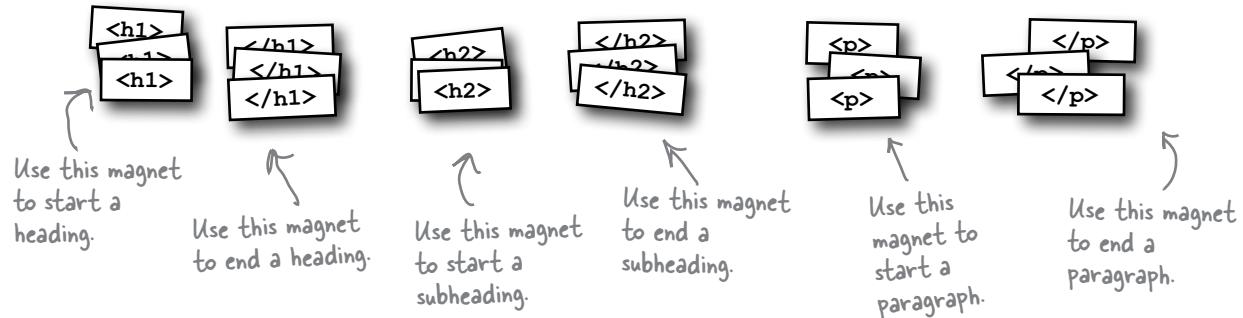
```
<h1> Starbuzz Coffee Beverages </h1>

House Blend, $1.49
A smooth, mild blend of coffees from Mexico, Bolivia and
Guatemala.

<h2> Mocha Cafe Latte, $2.35 </h2>
<p> Espresso, steamed milk and chocolate syrup. </p>

Cappuccino, $1.89
A mixture of espresso, steamed milk and foam.

Chai Tea, $1.85
A spicy drink made with black tea, spices, milk and honey.
```





Congratulations,
you've just written
your first HTML!

They might have looked like fridge magnets, but you were really *marking up* your text with HTML. Only, as you know, we usually refer to the magnets as *tags*.

Check out the markup below and compare it to your magnets on the previous page.

```
<h1>Starbuzz Coffee Beverages</h1>
```

Use the `<h1>` and `</h1>` tags to mark headings. All the text in between is the actual content of the heading.

```
<h2>House Blend, $1.49</h2>
```

```
<p>A smooth, mild blend of coffees from Mexico, Bolivia and Guatemala.</p>
```

```
<h2>Mocha Cafe Latte, $2.35</h2>
```

```
<p>Espresso, steamed milk and chocolate syrup.</p>
```

```
<h2>Cappuccino, $1.89</h2>
```

```
<p>A mixture of espresso, steamed milk and foam.</p>
```

```
<h2>Chai Tea, $1.85</h2>
```

```
<p>A spicy drink made with black tea, spices, milk and honey.</p>
```

The `<h2>` and `</h2>` tags go around a subheading. Think of an `<h2>` heading as a subheading of an `<h1>` heading.

The `<p>` and `</p>` tags go around a block of text that is a paragraph. That can be one or many sentences.

Notice that you don't have to put matching tags on the same line. You can put as much content as you like between them.

Are we there yet?

You have an HTML file with markup—does that make a web page? Almost. You've already seen the `<html>`, `<head>`, `<title>`, and `<body>` tags, and we just need to add those to make this a first-class HTML page...

First, surround your HTML with `<html>` & `</html>` tags. This tells the browser the content of the file is HTML.

```
<html>
```

```
<head>
    <title>Starbuzz Coffee</title>
</head>
```

Next add `<head>` and `</head>` tags. The head contains information about your web page, like its title. For now, think about it this way: the head allows you to tell the browser things about the web page.

Go ahead and put a title inside the head. The title usually appears at the top of the browser window.

The head consists of the `<head>` & `</head>` tags and everything in between.

```
<body>
```

```
<h1>Starbuzz Coffee Beverages</h1>
<h2>House Blend, $1.49</h2>
<p>A smooth, mild blend of coffees from Mexico,
    Bolivia and Guatemala.</p>

<h2>Mocha Cafe Latte, $2.35</h2>
<p>Espresso, steamed milk and chocolate syrup.</p>

<h2>Cappuccino, $1.89</h2>
<p>A mixture of espresso, steamed milk and foam.</p>

<h2>Chai Tea, $1.85</h2>
<p>A spicy drink made with black tea, spices,
    milk and honey.</p>
```

```
</body>
```

```
</html>
```

The body contains all the content and structure of your web page—the parts of the web page that you see in your browser.

Keep your head and body separate when writing HTML.



Another test drive



Go ahead and change your “index.html” file by adding in the `<head>`, `</head>`, `<title>`, `</title>`, `<body>` and `</body>` tags. Once you’ve done that, save your changes and reload the file into your browser.

You can reload the index.html file by selecting the Open File menu item again, or by using your browser’s reload button.

Notice that the title, which you specified in the `<head>` element, shows up here.

The screenshot shows a web browser window with the title "Starbuzz Coffee". The address bar indicates the file is located at "file:///chapter1/starbuzz/index.html". The main content area displays the following text:

Starbuzz Coffee Beverages

House Blend, \$1.49
A smooth, mild blend of coffees from Mexico, Bolivia and Guatemala.

Mocha Cafe Latte, \$2.35
Espresso, steamed milk and chocolate syrup.

Cappuccino, \$1.89
A mixture of espresso, steamed milk and foam.

Chai Tea, \$1.85
A spicy drink made with black tea, spices, milk and honey.

Now things look a bit better.
The browser has interpreted
your tags and created a
display for the page that
is not only more structured,
but also more readable.



Tags dissected

Okay, you've seen a bit of markup, so let's zoom in and take a look at how tags really work.



You usually put tags around some piece of content. Here we're using tags to tell the browser that our content, "Starbuzz Coffee Beverages," is a top-level heading (that is, heading level one).

Here's the opening tag that begins the heading.

Tags consist of the tag name surrounded by angle brackets; that is, the < and > characters.

<**h1**> Starbuzz Coffee Beverages </**h1**>

The whole shebang is called an element. In this case, we can call it the **<h1>** element. An element consists of the enclosing tags and the content in between.

This is the closing tag that ends the heading; in this case the </**h1**> tag is ending an **<h1>** heading. You know it's a closing tag because it comes after the content, and it's got a "/" before the "h1". All closing tags have a "/" in them.

We call an opening tag and its closing tag matching tags.

To tell the browser about the structure of your page, use pairs of tags around your content.

Remember:

Element = Opening Tag + Content + Closing Tag

^{there are no} Dumb Questions

Q: So matching tags don't have to be on the same line?

A: No; remember the browser doesn't really care about tabs, returns, and most spaces. So, your tags can start and end anywhere on the same line, or they can start and end on different lines. Just make sure you start with an opening tag, like `<h2>`, and end with a closing tag, like `</h2>`.

Q: Why do the closing tags have that extra "/"?

A: That "/" in the closing tag is to help both you and the browser know where a particular piece of structured content ends. Otherwise, the closing tags would look just like the opening tags, right?

Q: I've noticed the HTML in some pages doesn't always match opening tags with closing tags.

A: Well, the tags are supposed to match. In general, browsers do a pretty good job of figuring out what you mean if you write incorrect HTML. But, as you're going to see, these days there are big benefits to writing totally correct HTML. If you're worried you'll never be able to write perfect HTML, don't be; there are plenty of tools to verify your code before you put it on a web server so the whole world can see it. For now, just get in the habit of always matching your opening tags with closing tags.

Q: Well, what about that `` tag in the lounge example? Did you forget the closing tag?

A: Wow, sharp eye. There are some elements that use a shorthand notation with only one tag. Keep that in the back of your mind for now, and we'll come back to it in a later chapter.

Q: An element is an opening tag + content + closing tag, but can't you have tags inside other tags? Like the `<head>` and `<body>` are inside an `<html>` tag?

A: Yes, HTML tags are often "nested" like that. If you think about it, it's natural for an HTML page to have a body, which contains a paragraph, and so on. So many HTML elements have other HTML elements between their tags. We'll take a good look at this kind of thing in later chapters, but for now just get your mind noticing how the elements relate to each other in a page.



Tags can be a little more interesting than what you've seen so far. Here's the paragraph tag with a little extra added to it. What do you think this does?

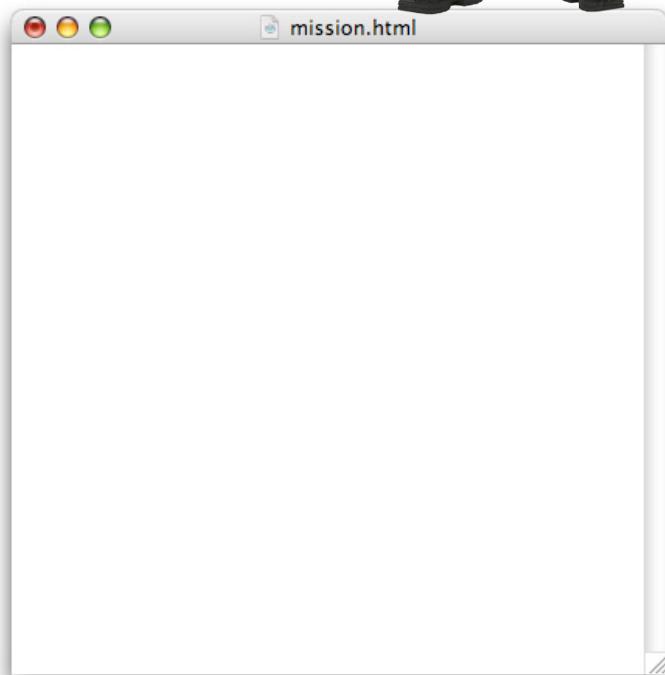
```
<p id="houseblend">A smooth, mild  
blend of coffees from Mexico, Bolivia  
and Guatemala.</p>
```



Exercise



Oh, I forgot to mention, we need our company mission on a page, too. Grab the mission statement off one of our coffee cups and create another page for it...



- ➊ Write the HTML for the new "mission.html" page here.
- ➋ Type in your HTML using a text editor, and save it as "mission.html" in the same folder as your "index.html" file.
- ➌ Once you've done that, open "mission.html" in your browser.
- ➍ Check your work at the end of the chapter before moving on...



Okay, it looks like you're getting somewhere. You've got the main page and the mission page all set. But don't forget the CEO said the site needs to look great too. Don't you think it needs a little style?

Right. We have the structure down, so now we're going to concentrate on its presentation.

You already know that HTML gives you a way to describe the structure of the content in your files. When the browser displays your HTML, it uses its own built-in default style to present this structure. But relying on the browser for style obviously isn't going to win you any "designer of the month" awards.

That's where CSS comes in. CSS gives you a way to describe how your content should be presented. Let's get our feet wet by creating some CSS that makes the Starbuzz page look a little more presentable (and launch your web career in the process).

CSS is an abbreviation for Cascading Style Sheets. We'll get into what that all means later, but for now just know that CSS gives you a way to tell the browser how elements in your page should look.

Meet the style element

To add style, you add a new (say it with us) E-L-E-M-E-N-T to your page—the `<style>` element. Let's go back to the main Starbuzz page and add some style. Check it out...

```

<html>
  <head>
    <title>Starbuzz Coffee</title>
    <style type="text/css">
      </style>
  </head>
  <body>
    <h1>Starbuzz Coffee Beverages</h1>

    <h2>House Blend, $1.49</h2>
    <p>A smooth, mild blend of coffees from Mexico, Bolivia and
Guatemala.</p>

    <h2>Mocha Caffe Latte, $2.35</h2>
    <p>Espresso, steamed milk and chocolate syrup.</p>

    <h2>Cappuccino, $1.89</h2>
    <p>A mixture of espresso, steamed milk and milk foam.</p>

    <h2>Chai Tea, $1.85</h2>
    <p>A spicy drink made with black tea, spices, milk and honey.</p>
  </body>
</html>

```

The `<style>` element is placed inside the head of your HTML.

Just like other elements, the `<style>` element has an opening tag, `<style>`, and a closing tag, `</style>`.

The `<style>` tag also has an (optional) attribute, called `type`, which tells the browser the kind of style you're using. Because you're going to use CSS, you can specify the "text/css" type.

And here's where you're going to define the styles for the page.

there are no Dumb Questions

Q: An element can have an “attribute”? What does that mean?

A: Attributes give you a way to provide additional information about an element. Like, if you have a `<style>` element, the attribute allows you to say exactly what kind of style you're talking about. You'll be seeing a lot more attributes for various elements; just remember they give you some extra info about the element.

Q: Why do I have to specify the type of the style (“text/css”) as an attribute of the style? Are there other kinds of style?

A: At one time the designers of HTML thought there would be other styles, but as it turns out we've all come to our senses since then and you can just use `<style>` without an attribute—all the browsers will know you mean CSS. We're disappointed; we were holding our breath for the `<style type="50sKitsch">` style. Oh well.

Giving Starbuzz some style...

Now that you've got a `<style>` element in the HTML head, all you need to do is supply some CSS to give the page a little pizzazz. Below you'll find some CSS already "baked" for you. Whenever you see the  logo, you're seeing HTML and CSS that you should type in as-is. *Trust us.* You'll learn how the markup works later, after you've seen what it can do.

So, take a look at the CSS and then add it to your "index.html" file. Once you've got it typed in, save your file.

```
<html>
  <head>
    <title>Starbuzz Coffee</title>
    <style type="text/css">
      body {
        background-color: #d2b48c;
        margin-left: 20%;
        margin-right: 20%;
        border: 2px dotted black;
        padding: 10px 10px 10px 10px;
        font-family: sans-serif;
      }
    </style>
  </head>

  <body>
    <h1>Starbuzz Coffee Beverages</h1>

    <h2>House Blend, $1.49</h2>
    <p>A smooth, mild blend of coffees from Mexico, Bolivia and Guatemala.</p>

    <h2>Mocha Caffe Latte, $2.35</h2>
    <p>Espresso, steamed milk and chocolate syrup.</p>

    <h2>Cappuccino, $1.89</h2>
    <p>A mixture of espresso, steamed milk and milk foam.</p>

    <h2>Chai Tea, $1.85</h2>
    <p>A spicy drink made with black tea, spices, milk and honey.</p>
  </body>
</html>
```



CSS uses a syntax that is totally different from HTML.

Cruisin' with style...



It's time for another test drive, so reload your "index.html" file again. This time, you'll see the Starbuzz web page has a whole new look.

Background color is now tan.

Now we have margins around the content

We've got a black dotted border around the content.

There's now some padding between the content and the border (on all sides).

We're using a different font for a cleaner look.

Starbuzz Coffee Beverages

House Blend, \$1.49

A smooth, mild blend of coffees from Mexico, Bolivia and Guatemala.

Mocha Caffe Latte, \$2.35

Espresso, steamed milk and chocolate syrup.

Cappuccino, \$1.89

A mixture of espresso, steamed milk and milk foam.

Chai Tea, \$1.85

A spicy drink made with black tea, spices, milk and honey.

Margin



Watch it!

If you're using IE, you might not see the border.

Internet Explorer does not display the border around the body correctly. Try loading the page in Firefox, Chrome or Safari to see the border.

Whoa! Very nice. We're in business now!



WHO DOES WHAT?

Even though you've just glanced at CSS, you've already begun to see what it can do. Match each line in the style definition to what it does.

`background-color: #d2b48c;`

Defines the font to use for text.

`margin-left: 20%;`

Defines a border around the body that is dotted and the color black.

`border: 2px dotted black;`

Sets the left and right margins to take up 20% of the page each.

`padding: 10px 10px 10px 10px;`

Sets the background color to tan.

`font-family: sans-serif;`

Creates some padding around the body of the page.

there are no Dumb Questions

Q: CSS looks like a totally different language than HTML. Why have two languages? That's just more for me to learn, right?

A: You are quite right that HTML and CSS are completely different languages, but that is because they have very different jobs. Just like you wouldn't use English to balance your checkbook, or math to write a poem, you don't use CSS to create structure or HTML to create style because that's not what they were designed for. While this does mean you need to learn two languages,

you'll discover that because each language is good at what it does, this is actually easier than if you had to use one language to do both jobs.

Q: #d2b48c doesn't look like a color. How is #d2b48c the color "tan"?

A: There are a few different ways to specify colors with CSS. The most popular is called a "hex code," which is what #d2b48c is. This really is a tan color. For now, just go with it, and we'll be showing you exactly how #d2b48c is a color a little later.

Q: Why is there a "body" in front of the CSS rules? What does that mean?

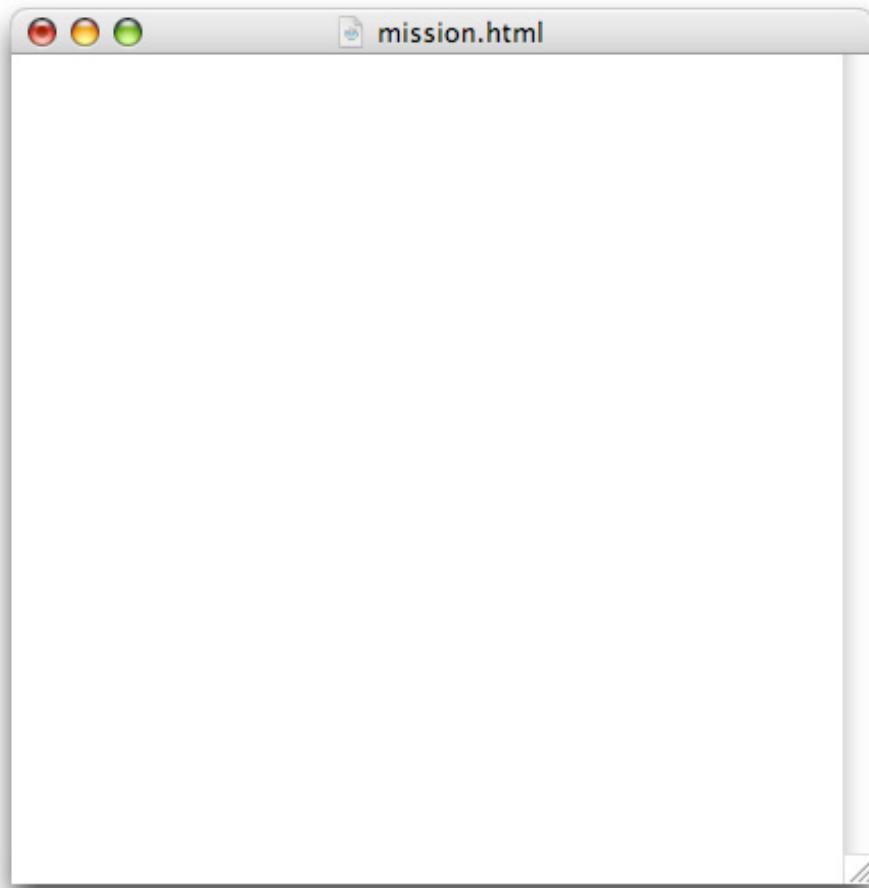
A: The "body" in the CSS means that all the CSS between the "{" and "}" applies to content within the HTML <body> element. So when you set the font to sans-serif, you're saying that the default font within the body of your page should be sans-serif.

We'll go into a lot more detail about how CSS works shortly, so keep reading. Soon, you'll see that you can be a lot more specific about how you apply these rules, and by doing so, you can create some pretty cool designs.



Now that you've put a little style in the Starbuzz "index.html" page, go ahead and update your "mission.html" page to have the same style.

- ➊ Write the HTML for the "mission.html" page below, and then add the new CSS.
- ➋ Update your "mission.html" file to include the new CSS.
- ➌ Once you've done that, reload "mission.html" in your browser.
- ➍ Make sure your mission page looks like ours at the end of the chapter.



Fireside Chats



Tonight's talk: **HTML and CSS on content and style**

HTML

Greetings, CSS; I'm glad you're here because I've been wanting to clear up some confusion about us.

Lots of people think that my tags tell the browsers how to *display* the content. It's just not true! I'm all about *structure*, not presentation.

Well, you can see how some people might get confused; after all, it's possible to use HTML without CSS and still get a decent-looking page.

Hey, I'm pretty powerful too. Having your content structured is much more important than having it look good. Style is so superficial; it's the structure of the content that matters.

Whoa, what an ego! Well, I guess I shouldn't expect anything else from you—you're just trying to make a fashion statement with all that style you keep talking about.

CSS

Really? What kind of confusion?

Heck yeah—I don't want people giving you credit for my work!

"Decent" might be overstating it a bit, don't you think? I mean, the way most browsers display straight HTML looks kinda crappy. People need to learn how powerful CSS is and how easily I can give their web pages great style.

Get real! Without me, web pages would be pretty damn boring. Not only that, but take away the ability to style pages and no one is going to take your pages seriously. Everything is going to look clumsy and unprofessional.

HTML

Right. In fact, we're totally different languages, which is good because I wouldn't want any of your style designers messing with my structure elements.

Yeah, that is obvious to me any time I look at CSS—talk about an alien language.

Millions of web writers would disagree with you. I've got a nice clean syntax that fits right in with the content.

Hey, ever heard of closing tags?

Just notice that no matter where you go, I've got you surrounded by <style> tags. Good luck escaping!

CSS

Fashion statement? Good design and layout can have a huge effect on how readable and usable pages are. And you should be happy that my flexible style rules allow designers to do all kinds of interesting things with your elements without messing up your structure.

Don't worry, we're living in separate universes.

Yeah, like HTML can be called a language? Who has ever seen such a clunky thing with all those tags?

Just take a look at CSS; it's so elegant and simple, no goofy angle brackets <around> <everything>. <See> <I> <can><talk> <just><like><Mr.><HTML><,><look><at> <me><!>

Ha! I'll show you...because, guess what? I *can* escape...

↑
Stay tuned!

Not only is this one fine cup of House Blend, but now we've got a web page to tell all our customers about our coffees. Excellent work. I've got some bigger ideas for the future; in the meantime, can you start thinking about how we are going to get these pages on the Internet so other people can see them?



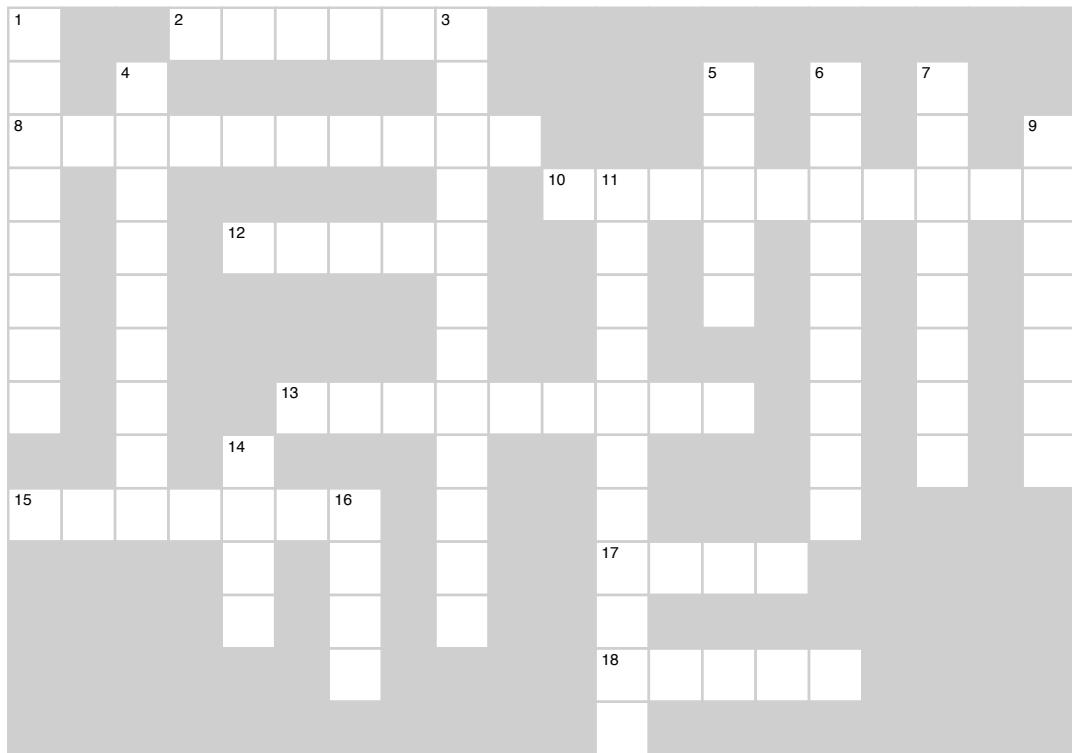
BULLET POINTS

- HTML and CSS are the languages we use to create web pages.
- Web servers store and serve web pages, which are created from HTML and CSS. Browsers retrieve pages and render their content based on the HTML and CSS.
- HTML is an abbreviation for HyperText Markup Language and is used to structure your web page.
- CSS is an abbreviation for Cascading Style Sheets, and is used to control the presentation of your HTML.
- Using HTML, we mark up content with tags to provide structure. We call matching tags, and their enclosed content, elements.
- An element is composed of three parts: an opening tag, content, and a closing tag. There are a few elements, like ``, that are an exception to this rule.
- Opening tags can have attributes. We've seen one already: type.
- Closing tags have a “/” after the left angle bracket, in front of the tag name, to distinguish them as closing tags.
- Your pages should always have an `<html>` element along with a `<head>` element and a `<body>` element.
- Information about the web page goes into the `<head>` element.
- What you put into the `<body>` element is what you see in the browser.
- Most whitespace (tabs, returns, spaces) is ignored by the browser, but you can use it to make your HTML more readable (to you).
- You can add CSS to an HTML web page by putting the CSS rules inside the `<style>` element. The `<style>` element should always be inside the `<head>` element.
- You specify the style characteristics of the elements in your HTML using CSS.



HTMLcross

It's time to sit back and give your left brain something to do. It's your standard crossword; all of the solution words are from this chapter.



Across

2. The "M" in HTML.
 8. Tags can have these to provide additional information.
 10. Browsers ignore this.
 12. You define presentation through this element.
 13. Purpose of <p> element.
 15. Two tags and content.
 17. What you see in your page.
 18. We emphasized Dance _____ Revolution.

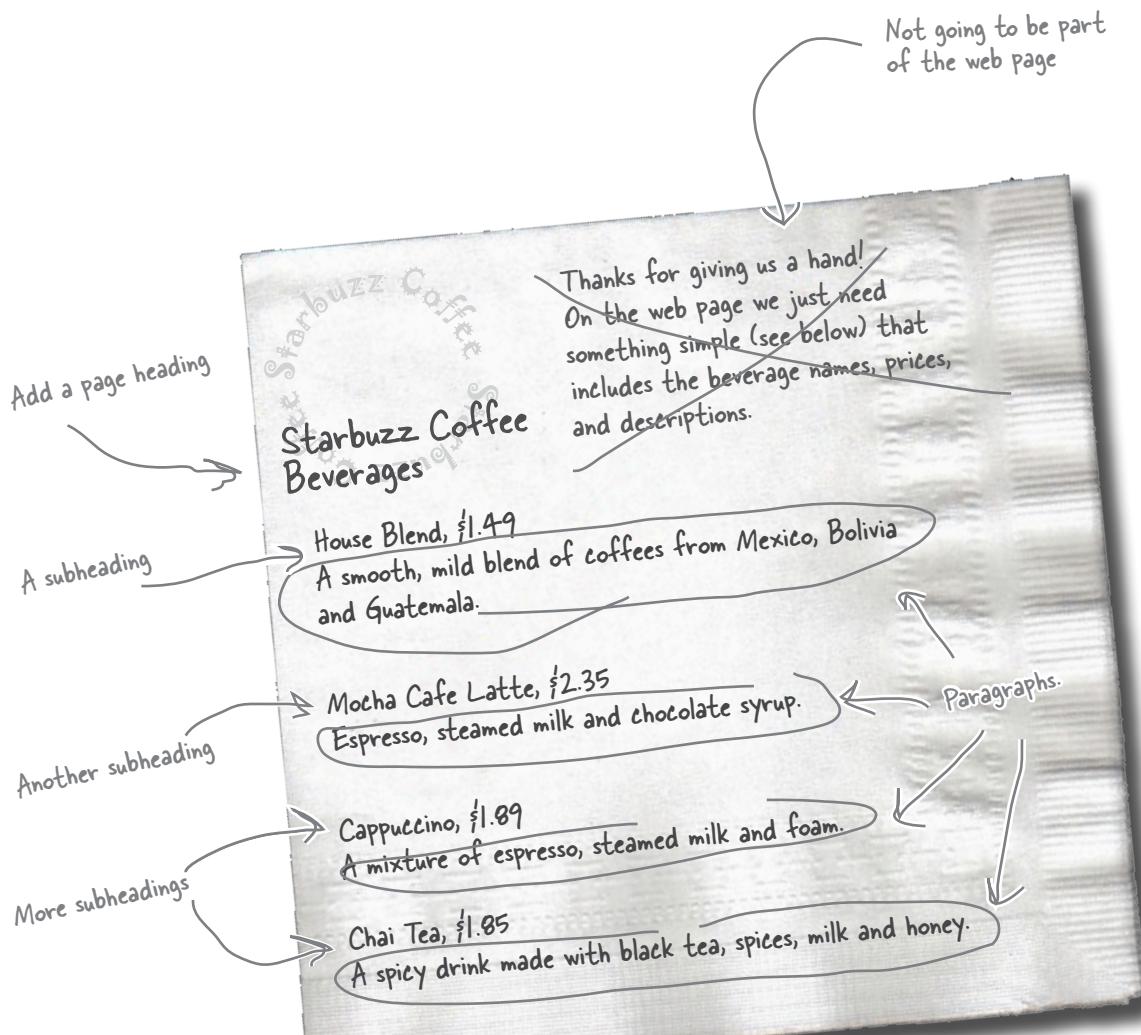
Down

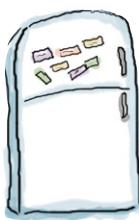
1. There are six of these.
 3. CSS is used when you need to control this.
 4. You mark up content to provide this.
 5. Appears at the top of the browser for each page.
 6. Style we wish we could have had.
 7. Company that launched your web career.
 9. Only type of style available.
 11. Always separate these in HTML.
 14. About your web page.
 16. Opening and closing.



Sharpen your pencil Solution

Go ahead and mark up the napkin (using your pencil) with any structure you see, and add anything that is missing.





Markup Magnets Solution

Your job was to add some structure to the text from the Starbuzz napkin. Use the fridge magnets at the bottom of the page to mark up the text so that you've indicated which parts are headings, subheadings, and paragraph text. Here's our solution.

```
<h1> Starbuzz Coffee Beverages </h1>

<h2> House Blend, $1.49 </h2>
<p> A smooth, mild blend of coffees from Mexico, Bolivia and
Guatemala. </p>

<h2> Mocha Cafe Latte, $2.35 </h2>
<p> Espresso, steamed milk and chocolate syrup. </p>

<h2> Cappuccino, $1.89 </h2>
<p> A mixture of espresso, steamed milk and foam. </p>

<h2> Chai Tea, $1.85 </h2>
<p> A spicy drink made with black tea, spices, milk and honey. </p>
```

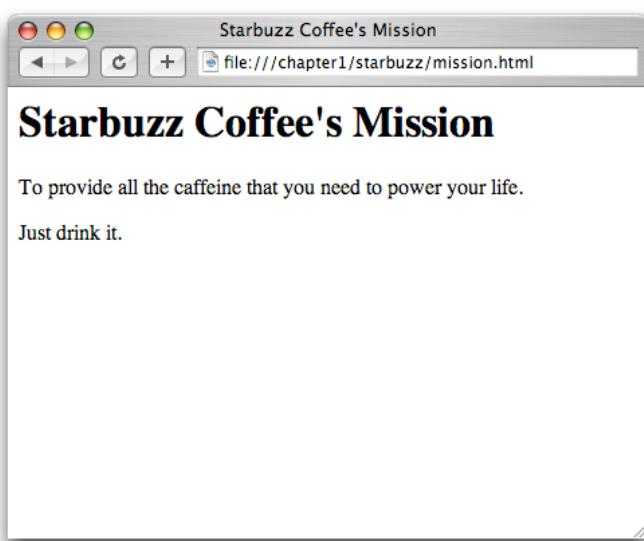
```
<h1>
<h1>
</h1>
</h1>
```



Leftover magnets



```
<html>
  <head>
    <title>Starbuzz Coffee's Mission</title>
  </head>
  <body>
    <h1>Starbuzz Coffee's Mission</h1>
    <p>To provide all the caffeine that you need to power your life.</p>
    <p>Just drink it.</p>
  </body>
</html>
```



Here's the HTML.

Here's the HTML displayed in a browser.



Exercise SOLUTION

Here's the CSS in the mission page.

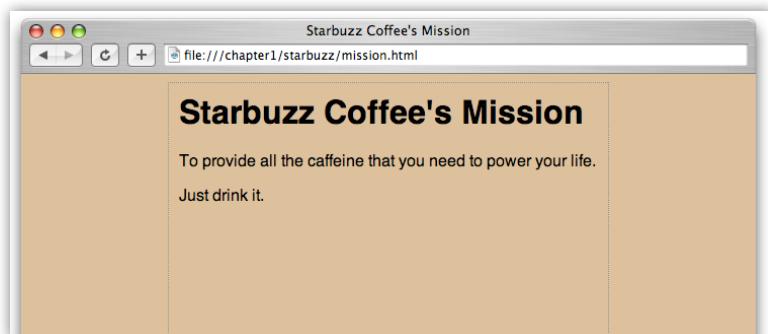


```

<html>
  <head>
    <title>Starbuzz Coffee's Mission</title>
    <style type="text/css">
      body {
        background-color: #d2b48c;
        margin-left: 20%;
        margin-right: 20%;
        border: 2px dotted black;
        padding: 10px 10px 10px 10px;
        font-family: sans-serif;
      }
    </style>
  </head>
  <body>
    <h1>Starbuzz Coffee's Mission</h1>
    <p>To provide all the caffeine that you need to power your life.</p>
    <p>Just drink it.</p>
  </body>
</html>

```

Now, the style matches the main Starbuzz page.





WHO DOES WHAT?

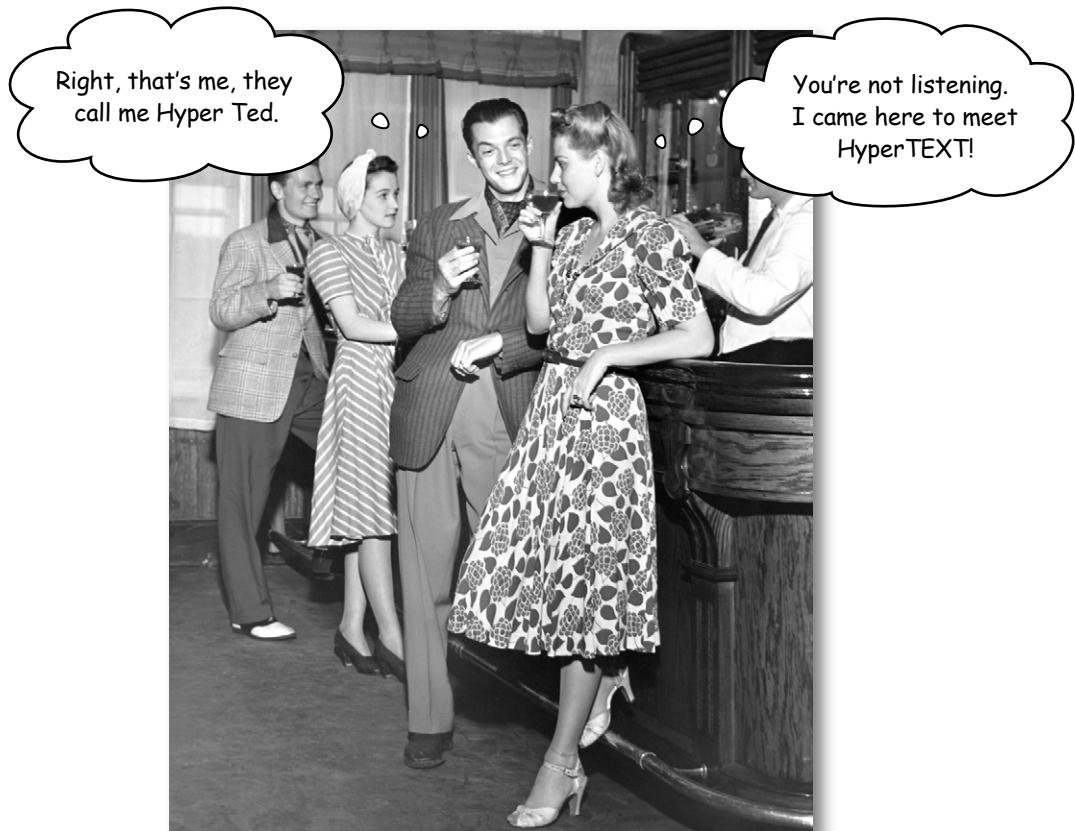
Even though you've just glanced at CSS, you've already seen the beginnings of what it can do. Match each line in the style definition to what it does.

- | | |
|---|--|
| <code>background-color: #d2b48c;</code> | Defines the font to use for text. |
| <code>margin-left: 20%;</code>
<code>margin-right: 20%;</code> | Defines a border around the body that is dotted and the color black. |
| <code>border: 2px dotted black;</code> | Sets the left and right margins to take up 20% of the page each. |
| <code>padding: 10px 10px 10px 10px;</code> | Sets the background color to tan. |
| <code>font-family: sans-serif;</code> | Creates some padding around the body of the page. |

'H		² M	A	R	K	U	³ P						
E		⁴ S				R			⁵ T	⁶ 5	⁷ S		
⁸ A	T	T	R	I	B	U	T	E	I	O	T	⁹ T	
D	R				S		¹⁰ W	¹¹ H	I	T	E		
I	U		¹² S	T	Y	L	E	E	L	K	R	X	
N	C				N		A	E	I	B	T		
G	T				T		D		T	U	C		
S	U		¹³ P	A	R	A	G	R	A	Z	S		
R		¹⁴ H		T		N			C	Z	S		
¹⁵ E	L	E	M	E	N	¹⁶ T	I	D		H			
									¹⁷ B	O	D	Y	
									O				
									¹⁸ D	A	N	C	E
									Y				

2 going further with hypertext

Meeting the “HT” in HTML



Did someone say “hypertext?” What’s that? Oh, only the entire basis of the Web. In Chapter 1 we kicked the tires of HTML and found it to be a nice markup language (the “ML” in HTML) for describing the structure of web pages. Now we’re going to check out the “HT” in HTML, hypertext, which will let us break free of a single page and link to other pages. Along the way we’re going to meet a powerful new element, the `<a>` element, and learn how being “relative” is a groovy thing. So, fasten your seat belts—you’re about to learn some hypertext.

Head First Lounge, new and improved

Remember the Head First Lounge? Great site, but wouldn't it be nice if customers could view a list of the refreshing elixirs? Even better, we should give customers some real driving directions so they can find the place.

The screenshot shows a web browser window titled "Head First Lounge" displaying the URL "file:///chapter2/completelounge/lounge.html". The page content includes:

- A heading: "Welcome to the New and Improved Head First Lounge".
- Four small images of colorful elixirs in glasses.
- A paragraph: "Join us any evening for refreshing [elixirs](#), conversation and maybe a game or two of *Dance Dance Revolution*. Wireless access is always provided; BYOWS (Bring Your Own Web Server)."
- A section header: "Directions".
- A paragraph: "You'll find us right in the center of downtown Webville. If you need help finding us, check out our [detailed directions](#). Come join us!"

Annotations with arrows point to specific elements:

- An arrow points from the text "Here's the new and improved page." to the browser window.
- An arrow points from the text "We've added links to two new pages, one for elixirs and one for driving directions." to the "elixirs" link in the main text.
- An arrow points from the text "The 'elixirs' link points to a page with a full list of elixir selections." to the "elixirs" link.
- An arrow points from the text "The 'detailed directions' link leads to an HTML page with driving directions." to the "detailed directions" link in the "Directions" section.
- An arrow points from the text "directions.html" to the "detailed directions" link.

Below the main browser window, a smaller window titled "Head First Lounge Directions" is shown, displaying the URL "file:///chapter2/completelounge/about/directions.html". The content of this window is:

```
Take the 305 S exit to Webville - go 0.4 mi
Continue on 305 - go 12 mi
Turn right at Structure Ave N - go 0.6 mi
Turn right and head toward Structure Ave N - go 0.0 mi
Turn right at Structure Ave N - go 0.7 mi
Continue on Structure Ave S - go 0.2 mi
Turn right at SW Presentation Way - go 0.0 mi
```

Head First Lounge Elixirs
file:///chapter2/completelounge/beverages/elixir.html

Our Elixirs

Green Tea Cooler



Chock full of vitamins and minerals, this elixir combines the healthful benefits of green tea with a twist of chamomile blossoms and ginger root.

Raspberry Ice Concentration



Combining raspberry juice with lemon grass, citrus peel and rosehips, this icy drink will make your mind feel clear and crisp.

Blueberry Bliss Elixir



Blueberries and cherry essence mixed into a base of elderflower herb tea will put you in a relaxed state of bliss in no time.

Cranberry Antioxidant Blast



Wake up to the flavors of cranberry and hibiscus in this vitamin C rich elixir.

A page listing some refreshing and healthy drinks. Feel free to grab one before going on.



elixir.html

Creating the new and improved lounge in three steps...

Let's rework the original Head First Lounge page so it links to the two new pages.

- 1** The first step is easy because we've already created the "directions.html" and "elixir.html" files for you. You'll find them in the source files for the book, which are available at <http://wickedlysmart.com/hfhtmlcss>. 
- 2** Next you're going to edit the "lounge.html" file and add in the HTML needed to link to "directions.html" and "elixir.html".
- 3** Last, you'll give the pages a test drive and try out your new links. When you get back, we'll sit down and look at how it all works.

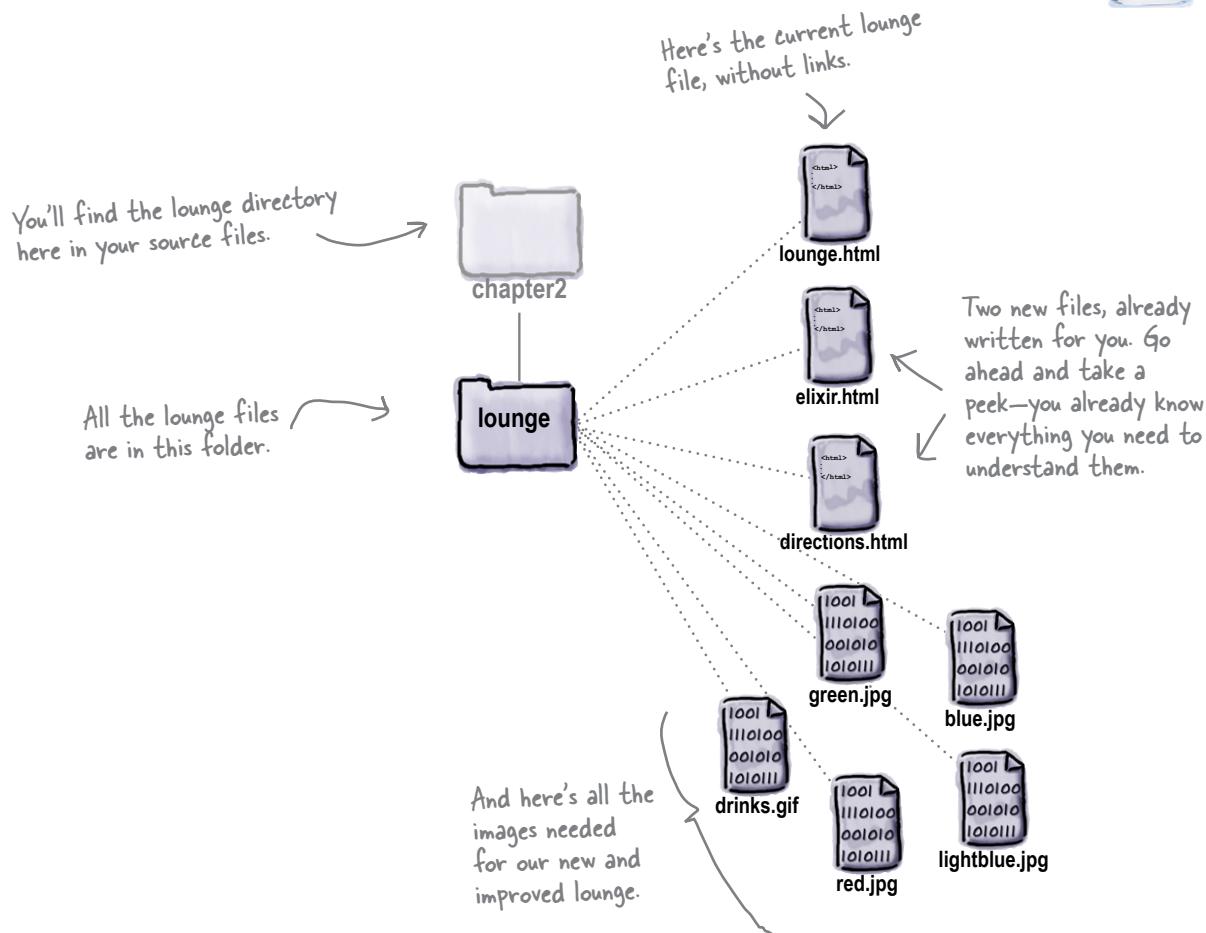
Flip the page and let's get started...

Creating the new lounge



1 Grab the source files

Go ahead and grab the source files from <http://wickedlysmart.com/hfhtmlcss>. Once you've downloaded them, look under the folder "chapter2/lounge" and you'll find "lounge.html", "elixir.html", and "directions.html" (and a bunch of image files).



The Head First Lounge is already growing; do you think that keeping all the site's files in a single directory is a good way to organize the site? What would you do differently?

2 Edit lounge.html

Open “lounge.html” in your editor. Add the new text and HTML that is highlighted below. Go ahead and type this in; we’ll come back and see how it all works on the next page.

```
<html>
  <head>
    <title>Head First Lounge</title>
  </head>
  <body>
    <h1>Welcome to the New and Improved Head First Lounge</h1>
    
    <p>
      Join us any evening for
      refreshing <a href="elixir.html">elixirs</a>,
      conversation and maybe a game or two of
      <em>Dance Dance Revolution</em>.
      Wireless access is always provided;
      BYOWS (Bring your own web server).
    </p>
    <h2>Directions</h2>
    <p>
      You'll find us right in the center of downtown Webville.
      If you need help finding us, check out
      our <a href="directions.html">detailed directions</a>.
      Come join us!
    </p>
  </body>
</html>
```

Let's add "New and Improved" to the heading.

Here's where we add the HTML for the link to the elixirs.

To create links, we use the `<a>` element; we'll take a look at how this element works in just a sec...

We need to add some text here to point customers to the new directions.

And here's where we add the link to the directions, again using an `<a>` element.

3 Save lounge.html and give it a test drive.

When you’re finished with the changes, save the file “lounge.html” and open it in your browser. Here are a few things to try:

- ➊ Click on the elixir link and the new elixir page will display.
- ➋ Click on the browser’s back button and “lounge.html” should be displayed again.
- ➌ Click on the directions link and the new directions page will display.

Behind
the Scenes



Okay, I've loaded the new lounge page, clicked the links, and everything worked. But I want to make sure I understand how the HTML works.



What did we do?

- ① Let's step through creating the HTML links. First, we need to put the text we want for the link in an `<a>` element, like this:

`<a>elixirs`

The `<a>` element is used to create a link to another page.

`<a>driving directions`

The content of the `<a>` element is the link text. In the browser, the link text appears with an underline to indicate you can click on it.

- ② Now that we have text for the link, we need to add some HTML to tell the browser where the link points to:

`elixirs`

The `href` attribute is how you specify the destination of the link.

For this link, the browser will display the text "elixirs" that, when clicked, will take the user to the "elixir.html" page.

`driving directions`

And for this link, the browser will display a "driving directions" link that, when clicked, will take the user to the "directions.html" page.

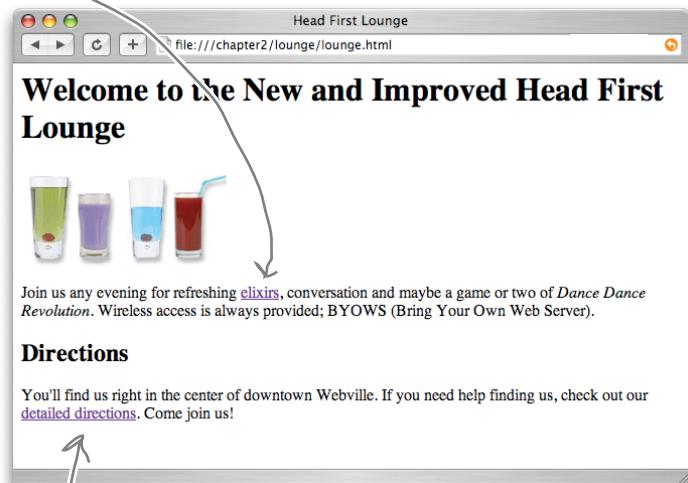
What does the browser do?

Behind the Scenes



- ① First, as the browser renders the page, if it encounters an `<a>` element, it takes the content of the element and displays it as a clickable link.

```
<a href="elixir.html">elixirs</a>
```



Both "elixirs" and "detailed directions" are between the opening and closing `<a>` tags, so they end up being clickable text in the web page.

```
<a href="directions.html">detailed directions</a>
```

Use the `<a>` element to create a hypertext link to another web page.
 The content of the `<a>` element becomes clickable in the web page.
 The `href` attribute tells the browser the destination of the link.

Behind the Scenes



- ② Next, when a user clicks on a link, the browser uses the "href" attribute to determine the page the link points to.

The user clicks on either the "elixirs" link or...



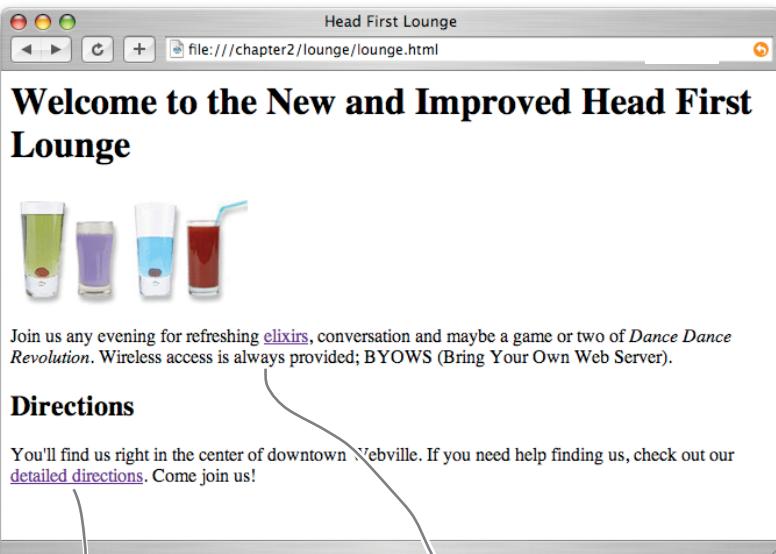
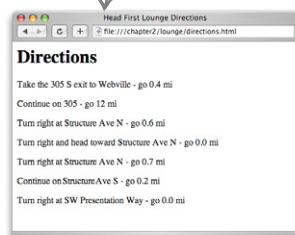
...on "detailed directions".



When "detailed directions" is clicked, the browser grabs the value of the href attribute, in this case "directions.html"...

`detailed directions`

...and loads "directions.html".



`elixirs`

...and displays the "elixir.html" page.



Understanding attributes

Attributes give you a way to specify additional information about an element. While we haven't looked at attributes in detail, you've already seen a few examples of them:

```
<style type="text/css">
<a href="irule.html">

```

The type attribute specifies which style language we're using, in this case CSS.

The href attribute tells us the destination of a hyperlink.

The src attribute specifies the filename of the picture an img tag displays.



Let's cook up an example to give you an even better feel for how attributes work:

What if <car> were an element?

If <car> were an element, then you'd naturally want to write some markup like this:

```
<car>My Red Mini</car>
```

With no attributes, all we can supply is a descriptive name for the car.

But this <car> element only gives a descriptive name for your car—it doesn't tell us the make, precise model, whether it is a convertible, or a zillion other details we might want to know. So, if <car> were really an element, we might use attributes like this:

```
<car make="Mini" model="Cooper" convertible="no">My Red Mini</car>
```

But with attributes, we can customize the element with all kinds of information.

Better, right? Now this markup tells us a lot more information in an easy-to-write, convenient form.



SAFETY FIRST

Attributes are always written the same way: first comes the attribute name, followed by an equals sign, and then the attribute value surrounded in double quotes.

You may see some sloppy HTML on the Web that leaves off the double quotes, but don't get lazy yourself. Being sloppy can cause you a lot of problems down the road (as we'll see later in the book).

Do this (best practice)

```
<a href="top10.html">Great Movies</a>
```

attribute name equals sign double quote attribute value double quote

Not this

```
<a href=top10.html>Great Movies</a>
```

No double quotes around the attribute value

*there are no
Dumb Questions*

Q: Can I just make up new attributes for an HTML element?

A: Web browsers only know about a predefined set of attributes for each element. If you just made up attributes, then browsers wouldn't know what to do with them, and as you'll see later in the book, doing this will very likely get you into trouble. When a browser recognizes an element or an attribute, we like to say that it "supports" that element or attribute. You should only use attributes that you know are supported.

That said, for programming web applications (the subject of *Head First HTML5 Programming*), HTML5 now supports custom data attributes that allow you to make up custom names for new attributes.

Q: Who decides what is "supported"?

A: There are standards committees that worry about the elements and attributes of HTML. These committees are made up of people ~~with nothing better to do~~ who generously give their time and energy to make sure there's a common HTML roadmap that all organizations can use to implement their browsers.

Q: How do I know what attributes and elements are supported? Or can all attributes be applied to any element?

A: Only certain attributes can be used with a given element. Think about it this way: you wouldn't use an attribute "convertible" with the element <toaster>, would you? So, you only want to use attributes that make sense and are supported by the element.

You're going to be learning which attributes are supported by which elements as you make your way through the book. After you've finished the book, there are lots of great references you can use to refresh your memory, such as *HTML & XHTML: The Definitive Guide* (O'Reilly).





Attributes Exposed

This week's interview:
Confessions of the href attribute

Head First: Welcome, href. It's certainly a pleasure to interview as big an attribute as you.

href: Thanks. It's good to be here and get away from all the linking; it can wear an attribute out. Every time someone clicks on a link, guess who gets to tell the browser where to go next? That would be me.

Head First: We're glad you could work us into your busy schedule. Why don't you take us back to the beginning...What does it mean to be an attribute?

href: Sure. Well, attributes are used to customize an element. It's easy to wrap some `<a>` tags around a piece of content, like "Sign up now!"—we do it like this: `<a>Sign up now!`—but without me, the href attribute, you have no way to tell the `<a>` element the destination of the link.

Head First: Got it so far...

href: ...but with an attribute you can provide additional information about the element. In my case, that's where the link points to:
`Sign up now!`. This says that the `<a>` element, which is labeled "Sign up now!", links to the "signup.html" page. Now, there are lots of other attributes in the world, but I'm the one you use with the `<a>` element to tell it where it points to.

Head First: Nice. Now, I have to ask, and I hope you aren't offended, but what is with the name? href? What's with that?

href: It's an old Internet family name. It means "hypertext reference," but all my friends just call me "href" for short.

Head First: Which is?

href: A hypertext reference is just another name for a resource that is on the Internet or your computer. Usually the resource is a web page, but I can also point to PDF documents...all kinds of things.

Head First: Interesting. All our readers have seen so far are links to their own pages; how do we link to other pages and resources on the Web?

href: Hey, I gotta get back to work, the whole Web is getting gunked up without me. Besides, isn't it your job to teach them this stuff?

Head First: Okay, okay, yes, we're getting to that in a bit...thanks for joining us, href.



You've created links to go from "lounge.html" to "elixir.html" and "directions.html"; now we're going to go back the other way. Below you'll find the HTML for "elixir.html". Add a link with the label "Back to the Lounge" at the bottom of the elixir page that points back to "lounge.html".

```
<html>
  <head>
    <title>Head First Lounge Elixirs</title>
  </head>
  <body>
    <h1>Our Elixirs</h1>

    <h2>Green Tea Cooler</h2>
    <p>
      
      Chock full of vitamins and minerals, this elixir
      combines the healthful benefits of green tea with
      a twist of chamomile blossoms and ginger root.
    </p>
    <h2>Raspberry Ice Concentration</h2>
    <p>
      
      Combining raspberry juice with lemon grass,
      citrus peel and rosehips, this icy drink
      will make your mind feel clear and crisp.
    </p>
    <h2>Blueberry Bliss Elixir</h2>
    <p>
      
      Blueberries and cherry essence mixed into a base
      of elderflower herb tea will put you in a relaxed
      state of bliss in no time.
    </p>
    <h2>Cranberry Antioxidant Blast</h2>
    <p>
      
      Wake up to the flavors of cranberry and hibiscus
      in this vitamin C rich elixir.
    </p>
  </body>
</html>
```

Your new
HTML goes ↗
here.

When you are done, go ahead and do the same with "directions.html" as well.



We need some help constructing and deconstructing `<a>` elements. Given your new knowledge of the `<a>` element, we're hoping you can help. In each row below, you'll find some combination of the label, destination, and the complete `<a>` element. Fill in any information that is missing. The first row is done for you.

Label	Destination	What you write in HTML
Hot or Not?	hot.html	<code>Hot or Not?</code>
Resume	cv.html	
	candy.html	<code>Eye Candy</code>
See my mini	mini-cooper.html	
let's play		<code>.....</code>

there are no Dumb Questions

Q: I've seen many pages where I can click on an image rather than text. Can I use the `<a>` element for that?

A: Yes, if you put an `` element between the `<a>` tags, then your image will be clickable just like text. We're not going to talk about images in depth for a few chapters, but they work just fine as links.

Q: So I can put anything between the `<a>` tags and it will be clickable? Like, say, a paragraph?

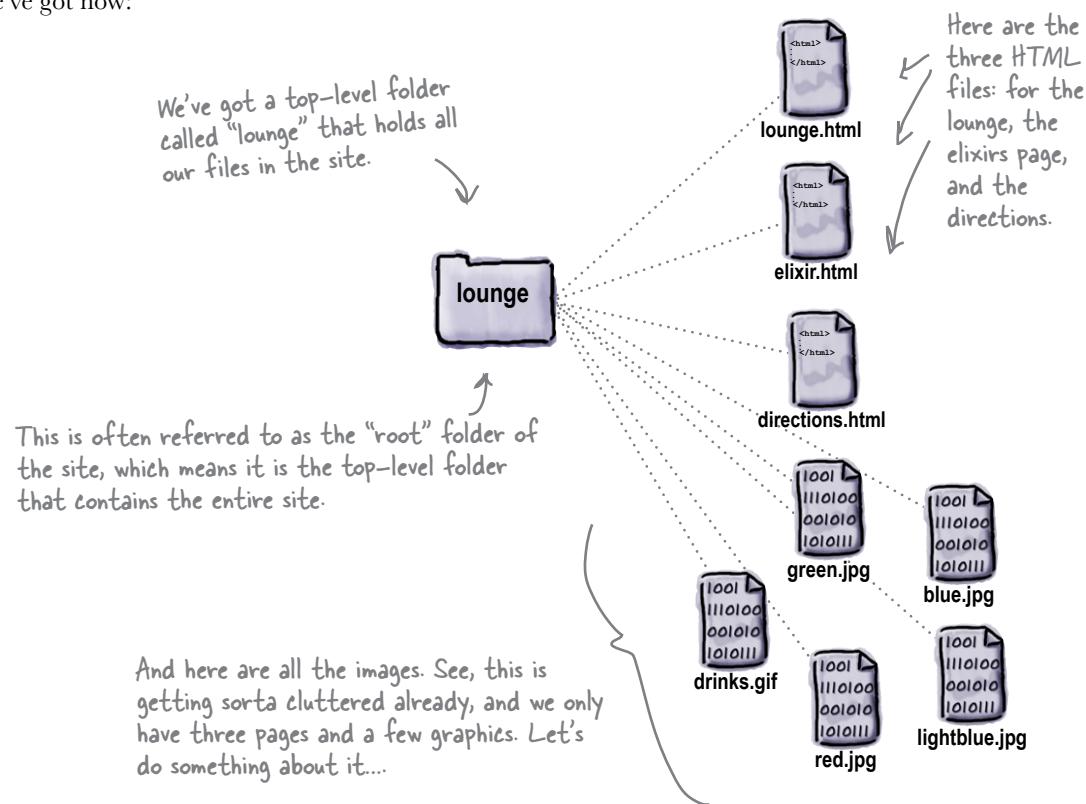
A: You can indeed put a `<p>` element inside an `<a>` element to link an entire paragraph. You'll mostly be using text and images (or both) within the `<a>` element, but if you need to link a `<p>` or a `<h1>` element, you can. What tags will go inside other tags is a whole other topic, but don't worry; we'll get there soon enough.



Getting organized

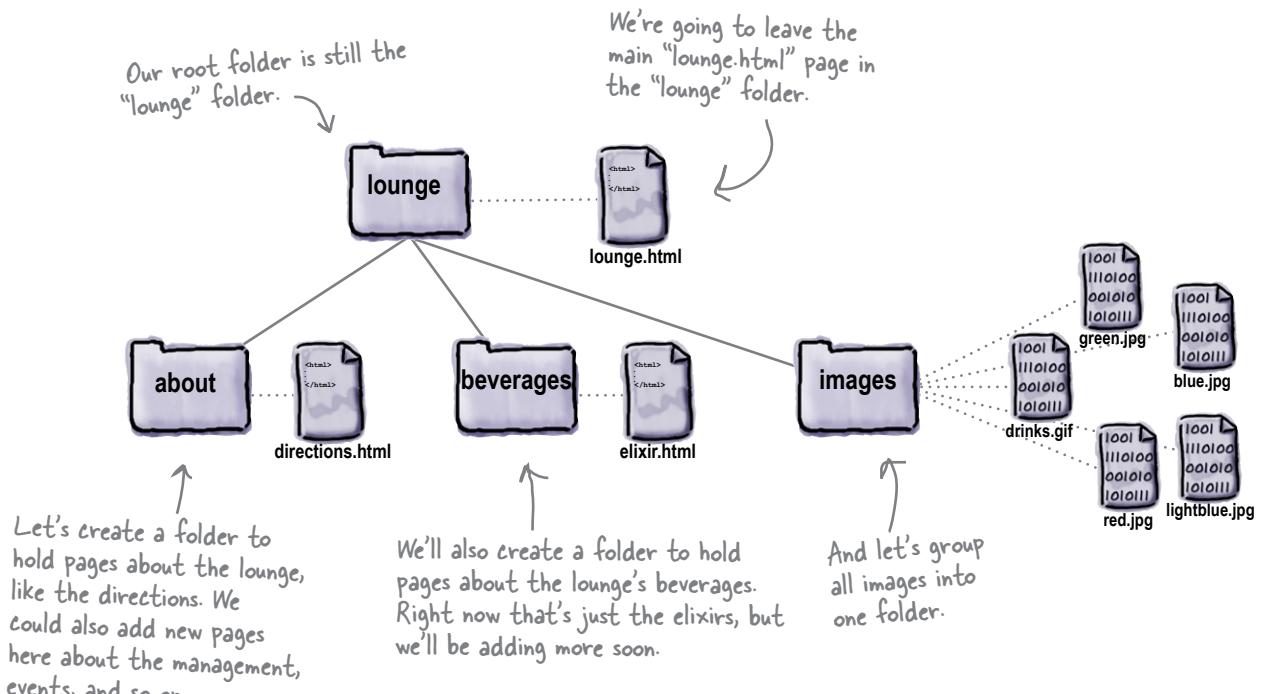
Before you start creating more HTML pages, it's time to get things organized. So far, we've been putting all our files and images in one folder. You'll find that even for modestly sized websites, things are much more manageable if you organize your web pages, graphics, and other resources into a set of folders.

Here's what we've got now:



Organizing the lounge...

Let's give the lounge site some meaningful organization now. Keep in mind there are lots of ways to organize any site; we're going to start simple and create a couple of folders for pages. We'll also group all those images into one place.



there are no Dumb Questions

Q: Since you have a folder for images, why not have another one called “html” and put all the HTML in that folder?

A: You could. There aren't any “correct” ways to organize your files; rather, you want to organize them in a way that works best for you and your users. As with most design decisions, you want to choose an organization scheme that is flexible enough to grow, while keeping things as simple as you can.

Q: Or why not put an images folder in each other folder, like “about” and “beverages”?

A: Again, we could have. We expect that some of the images will be reused among several pages, so we put images in a folder at the root (the top level) to keep them all together. If you have a site that needs lots of images in different parts of the site, you might want each branch to have its own image folder.

Q: “Each branch”?

A: You can understand the way folders are described by looking at them as upside down trees. At the top is the root and each path down to a file or folder is a branch.





Now you need to create the file and folder structure shown on the previous page. Here's exactly what you need to do:

- ❶ Locate your "lounge" folder and create three new subfolders named "about", "beverages", and "images".
- ❷ Move the file "directions.html" into the "about" folder.
- ❸ Move the file "elixir.html" into the "beverages" folder.
- ❹ Move all the images into the "images" folder.
- ❺ Finally, load your "lounge.html" file and try out the links.
Compare with how ours worked below.

Technical difficulties

It looks like we've got a few problems with the lounge page after moving things around.

We've got an image that isn't displaying. We usually call this a "broken image."

And, when you click on "elixirs" (or "detailed directions") things get much worse: we get an error saying the page can't be found.

The browser window shows the URL `file:///~/headfirst/lounge/lounge.html`. The page content is:

Welcome to the New and Improved Head First Lounge

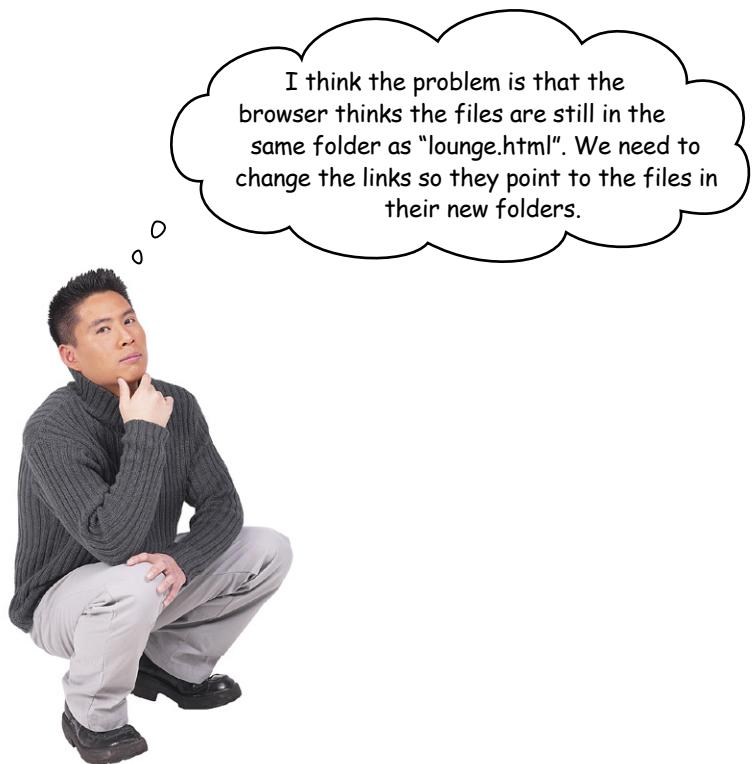
Join us any evening for refreshing [elixirs](#), conversation and maybe a game or two of *Dance Dance Revolution*. Wireless access is always provided; BYOWS (Bring your own web server).

Directions

You'll find us right in the center of downtown Webville. If you need help finding us, check out our [detailed directions](#). Come join us!

The alert dialog box has a yellow warning icon and the message: "The file /C:/lounge/elixir.html cannot be found. Please check the location and try again." A button labeled "OK" is at the bottom.

Some browsers display this error as a web page rather than a dialog box.



I think the problem is that the browser thinks the files are still in the same folder as "lounge.html". We need to change the links so they point to the files in their new folders.

Right. We need to tell the browser the new location of the pages.

So far you've used `href` values that point to pages in the *same folder*. Sites are usually a little more complicated, though, and you need to be able to point to pages that are in *other folders*.

To do that, you trace the path from your page to the destination file. That might mean going down a folder or two, or up a folder or two, but either way we end up with a *relative path* that we can put in the `href`.

Planning your paths...

What do you do when you're planning that vacation in the family truckster? You get out a map and start at your current location, and then trace a path to the destination. The directions themselves are *relative* to your location—if you were in another city, they'd be different directions, right?

To figure out a relative path for your links, it's the same deal: you start from the page that has the link, and then you trace a path through your folders until you find the file you need to point to.

Let's work through a couple of relative paths (and fix the lounge at the same time).

Okay, you'd really go to Google Maps, but work with us here!

There are other kinds of paths too. We'll get to those in later chapters.



Linking down into a subfolder

① Linking from "lounge.html" to "elixir.html".

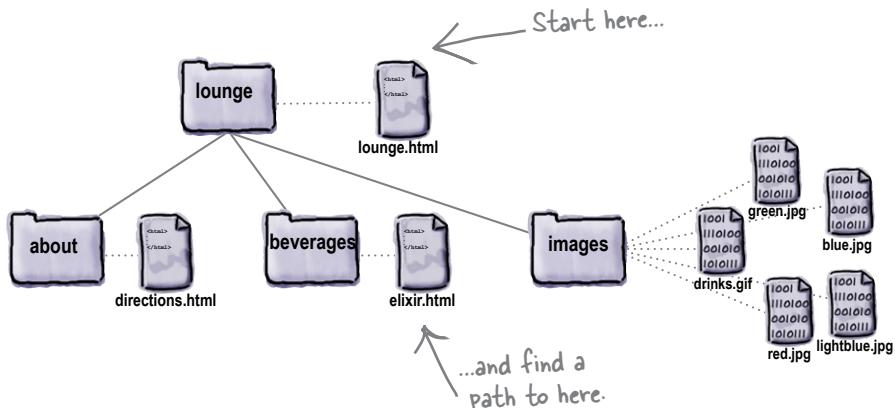
We need to fix the “elixirs” link in the “lounge.html” page. Here’s what the `<a>` element looks like now:

```
<a href="elixir.html">elixirs</a>
```

Right now we’re just using the filename “elixir.html”, which tells the browser to look in the same folder as “lounge.html”.

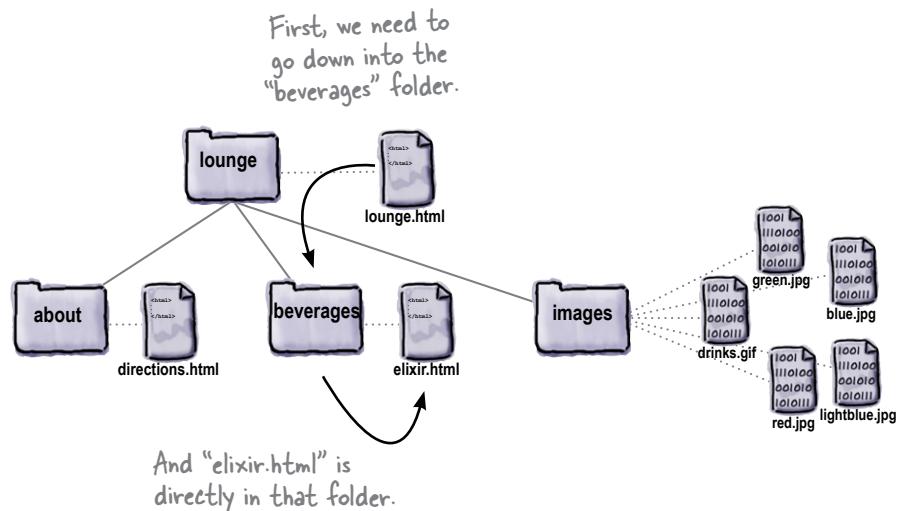
② Identify the source and the destination.

When we reorganized the lounge, we left “lounge.html” in the “lounge” folder, and we put “elixir.html” in the “beverages” folder, which is a subfolder of “lounge”.



③ Trace a path from the source to the destination.

Let's trace the path. To get from the "lounge.html" file to "elixir.html", we need to go into the "beverages" folder first, and then we'll find "elixir.html" in that folder.



④ Create an href to represent the path we traced.

Now that we know the path, we need to get it into a format the browser understands. Here's how you write the path:

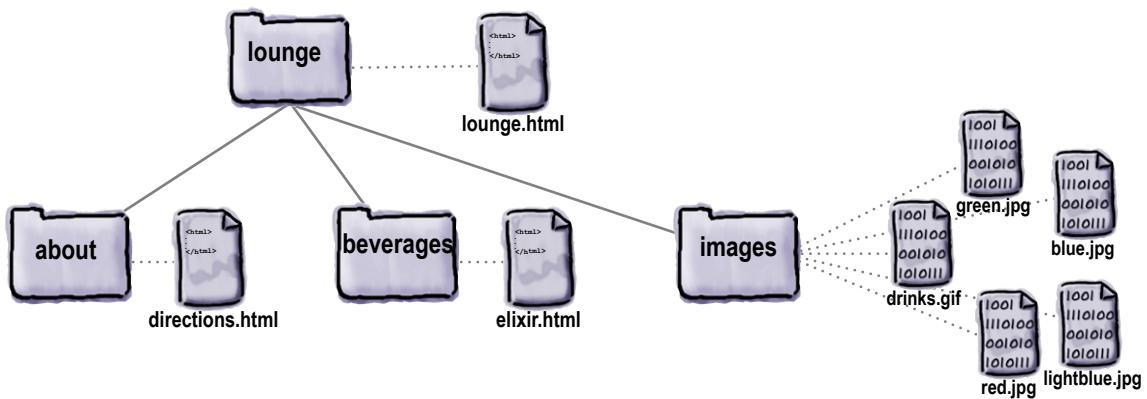


We put the relative path into the href value. Now when the link is clicked, the browser will look for the "elixir.html" file in the "beverages" folder.



Sharpen your pencil

Your turn: trace the relative path from “lounge.html” to “directions.html”. When you’ve discovered it, complete the `<a>` element below. Check your answer in the back of the chapter, and then go ahead and change both `<a>` elements in “lounge.html.”



`detailed directions`

YOUR ANSWER HERE ↗

Going the other way; linking up into a “parent” folder

- ① Linking from “directions.html” to “lounge.html”.

Now we need to fix those “Back to the Lounge” links. Here’s what the `<a>` element looks like in the “directions.html” file:

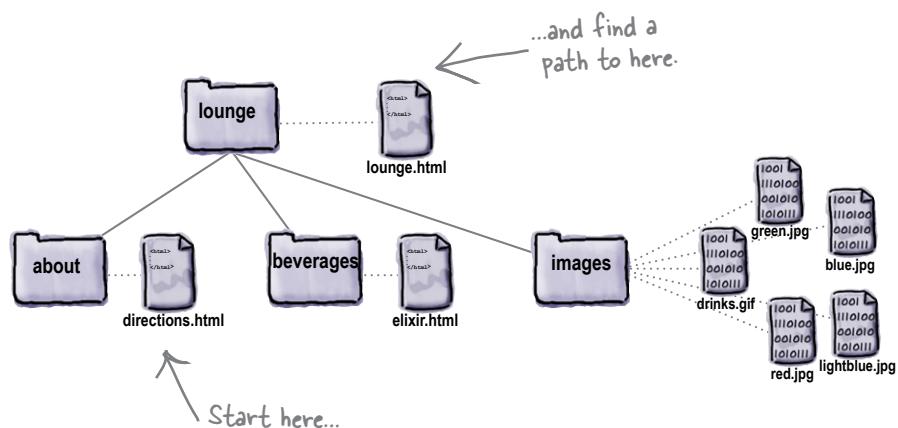
```
<a href="lounge.html">Back to the Lounge</a>
```

Right now, we’re just using the filename “lounge.html”, which tells the browser to look in the same folder as “directions.html”. That’s not going to work.

- ② Identify the source and the destination.

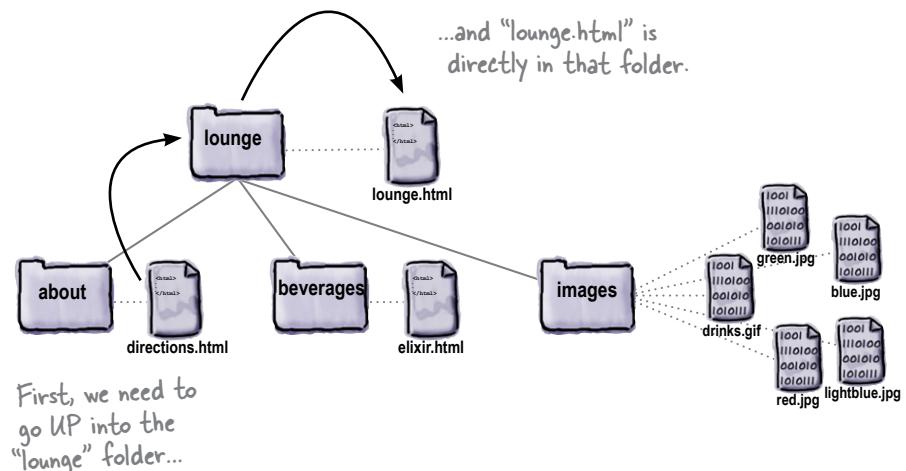
Let’s take a look at the source and destination.

The source is now the “directions.html” file, which is down in the “about” folder. The destination is the “lounge.html” file that sits above the “about” folder, where “directions.html” is located.



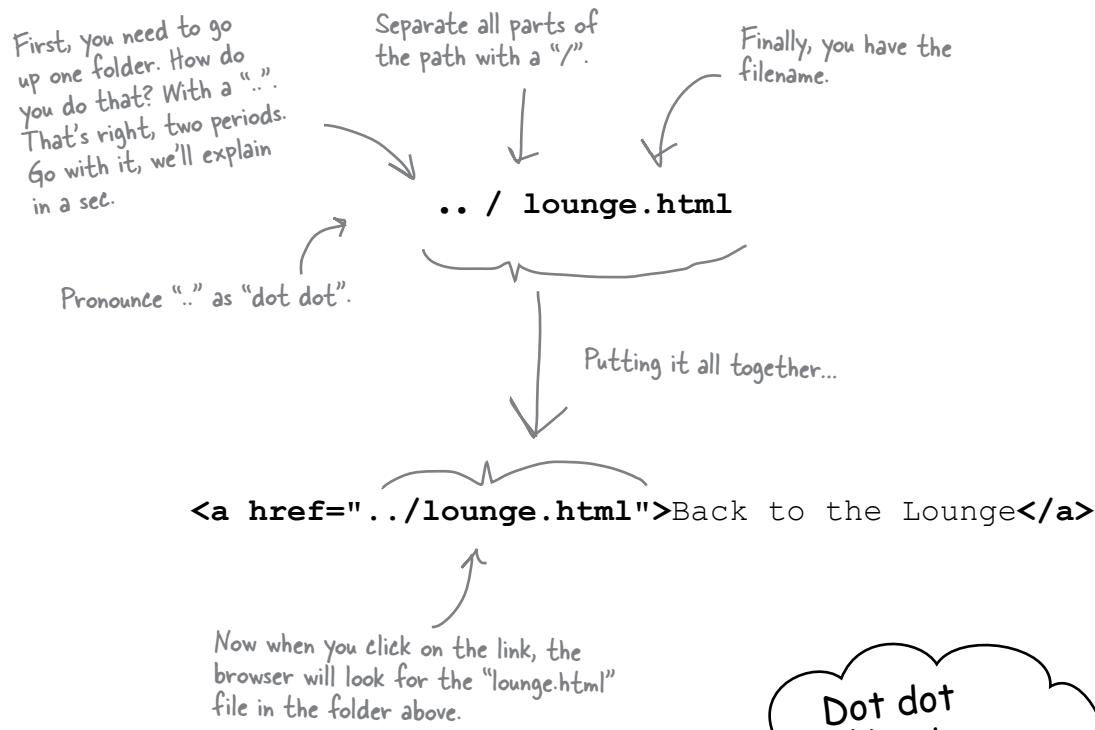
- ③ Trace a path from the source to the destination.

Let’s trace the path. To get from the “directions.html” file to “lounge.html”, we need to go up one folder into the “lounge” folder, and then we’ll find “lounge.html” in that folder.



④ Create an href to represent the path we traced.

We're almost there. Now that you know the path, you need to get it into a format the browser understands. Let's work through this:



Dot dot
Up, down,
housewares,
lingerie?



there are no Dumb Questions

Q: What's a parent folder? If I have a folder "apples" inside a folder "fruit", is "fruit" the parent of "apples"?

A: Exactly. Folders (you might have heard these called directories) are often described in terms of family relationships. For instance, using your example, "fruit" is the parent of "apples", and "apples" is the child of "fruit". If you had another folder "pears" that was a child of "fruit", it would be a sibling of "apples." Just think of a family tree.

Q: Okay, parent makes sense, but what is "..."?

A: When you need to tell the browser that the file you're linking to is in the parent folder, you use "..." to mean "move UP to the parent folder." In other words, it's browser-speak for parent.

In our example, we wanted to link from "directions.html", which is in the "about" folder, to "lounge.html", which is in the "lounge" folder, the parent of "about". So we had to tell the browser to look UP one folder, and "..." is the way we tell the browser to go UP.

Q: What do you do if you need to go up two folders instead of just one?

A: You can use ".." for each parent folder you want to go up. Each time you use ".." you're going up by one parent folder. So, if you want to go up two folders, you'd type "...". You still have to separate each part with the "/", so don't forget to do that (the browser won't know what "...." means!).

Q: Once I'm up two folders, how do I tell the browser where to find the file?

A: You combine the "..." with the filename. So, if you're linking to a file called "fruit.html" in a folder that's two folders up, you'd write "../fruit.html". You might expect that we'd call "..." the "grandparent" folder, but we don't usually talk about them that way, and instead say, "the parent of the parent folder," or "..." for short.

Q: Is there a limit to how far up I can go?

A: You can go up until you're at the root of your website. In our example, the root was the "lounge" folder. So, you could only go up as far as "lounge".

Q: What about in the other direction—is there a limit to how many folders I can go down?

A: Well, you can only go down as many folders as you have created. If you create folders that are 10 deep, then you can write a path that takes you down 10 folders. But we don't recommend that—when you have that many folder levels, it probably means your website organization is too complicated!

In addition, some browsers impose a limit on the number of characters you can have in a path. The spec advises caution above 255 characters, although modern browsers support longer lengths. If you have a large site, however, it's something to be aware of.

Q: My operating system uses "\ as a separator; shouldn't I be using that instead of "/"?

A: No; in web pages you always use "/" (forward slash). Don't use "\ (backslash). Various operating systems use different file separators (for instance, Windows uses "\ instead of "/") but when it comes to the Web, we pick a common separator and all stick to it. So, whether you're using Mac, Windows, Linux, or something else, always use "/" in the paths in your HTML.



Your turn: trace the relative path from "elixir.html" to "lounge.html" from the "Back to the Lounge" link. How does it differ from the same link in the "directions.html" file?

Answer: It doesn't; it is exactly the same.

Fixing those broken images...

You've almost got the lounge back in working order; all you need to do now is fix those images that aren't displaying.

We haven't looked at the `` element in detail yet (we will in a couple of chapters), but all you need to know for now is that the `` element's `src` attribute takes a relative path, just like the `href` attribute.

Here's the image element from the "lounge.html" file:

```

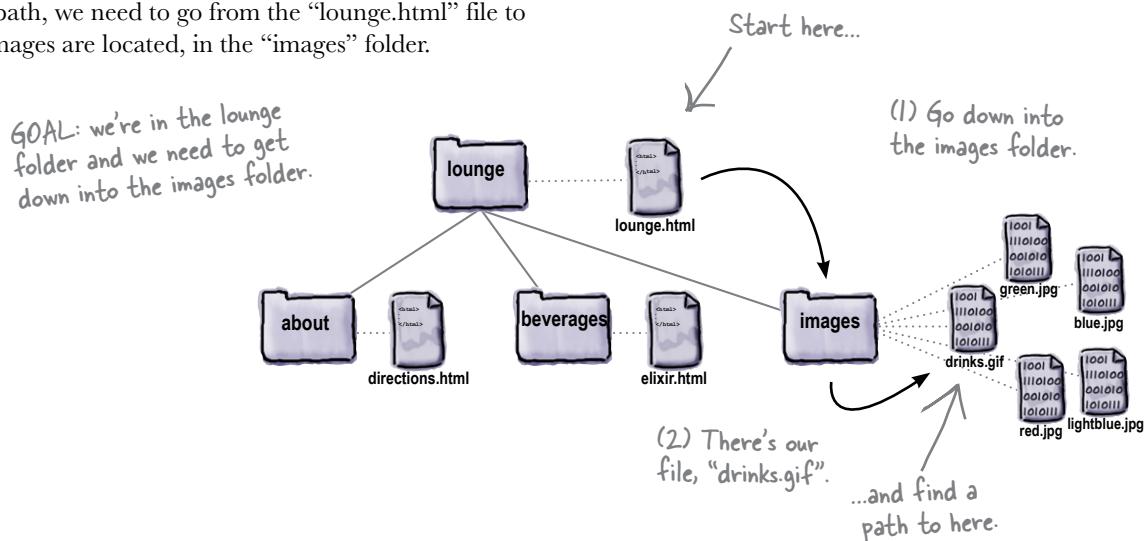
```

Here's the relative path, which tells the browser where the image is located. We specify this just like we do with the `href` attribute in the `<a>` element.



Finding the path from "lounge.html" to "drinks.gif"

To find the path, we need to go from the "lounge.html" file to where the images are located, in the "images" folder.



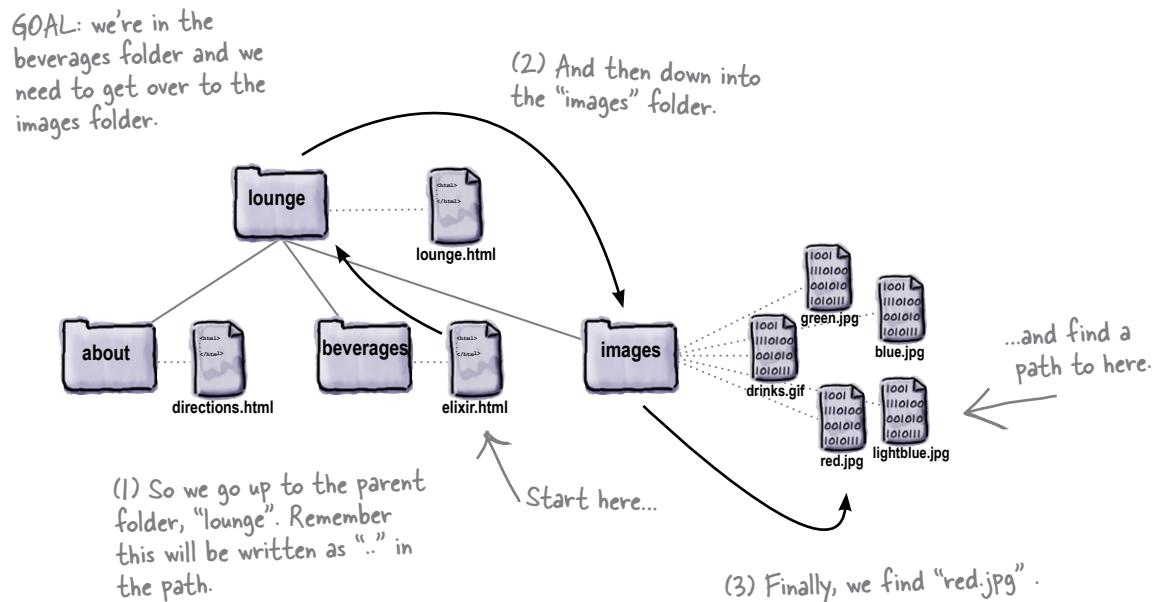
So when we put (1) and (2) together, our path looks like "images/drinks.gif", or:

```

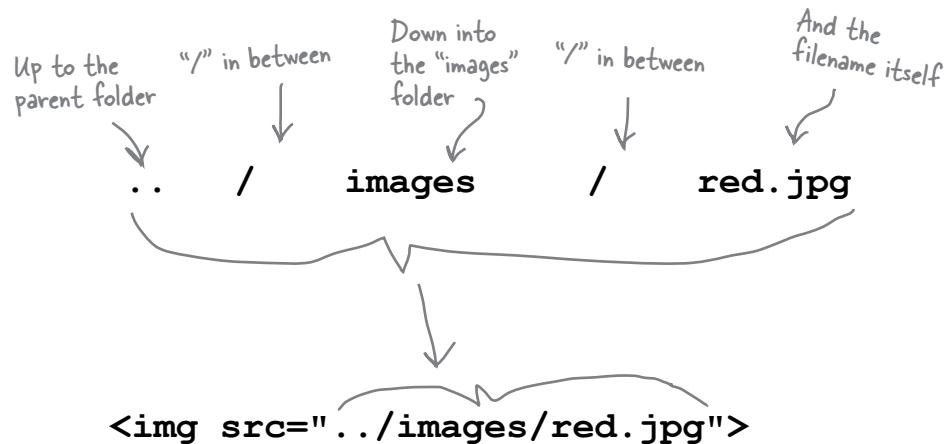
```

Finding the path from "elixir.html" to "red.jpg"

The elixirs page contains images of several drinks: "red.jpg", "green.jpg", "blue.jpg", and so on. Let's figure out the path to "red.jpg" and then the rest will have a similar path because they are all in the same folder:



So putting (1), (2), and (3) together, we get:



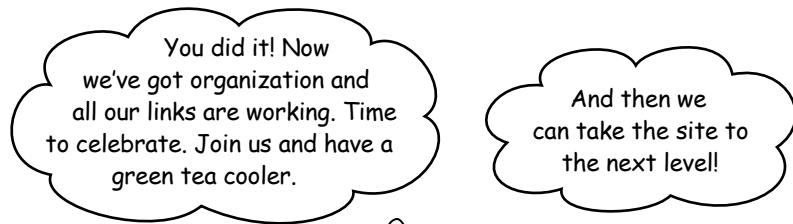
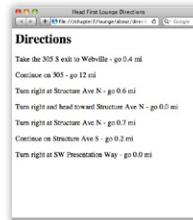


That covers all the links we broke when we reorganized the lounge, although you still need to fix the images in your “lounge.html” and “elixir.html” files. Here’s exactly what you need to do:

- ❶ In “lounge.html”, update the image `src` attribute to have the value “images/drinks.gif”.
- ❷ In “elixir.html”, update the image `src` attribute so that “./images/” comes before each image name.
- ❸ Save both files and load “lounge.html” in your browser. You’ll now be able to navigate between all the pages and view the images.



P.S. If you’re having any trouble, the folder “chapter2/completelounge” contains a working version of the lounge. Double-check your work against it.





BULLET POINTS

- When you want to link from one page to another, use the `<a>` element.
- The `href` attribute of the `<a>` element specifies the destination of the link.
- The content of the `<a>` element is the label for the link. The label is what you see on the web page. By default, it's underlined to indicate you can click on it.
- You can use words or an image as the label for a link.
- When you click on a link, the browser loads the web page that's specified in the `href` attribute.
- You can link to files in the same folder, or files in other folders.
- A relative path is a link that points to other files on your website relative to the web page you're linking from. Just like on a map, the destination is relative to the starting point.
- Use “..” to link to a file that's one folder above the file you're linking from.
- “..” means “parent folder.”
- Remember to separate the parts of your path with the “/” (forward slash) character.
- When your path to an image is incorrect, you'll see a broken image on your web page.
- Don't use spaces in the names you choose for files and folders for your website.
- It's a good idea to organize your website files early on in the process of building your site, so you don't have to change a bunch of paths later when the website grows.
- There are many ways to organize a website; how you do it is up to you.



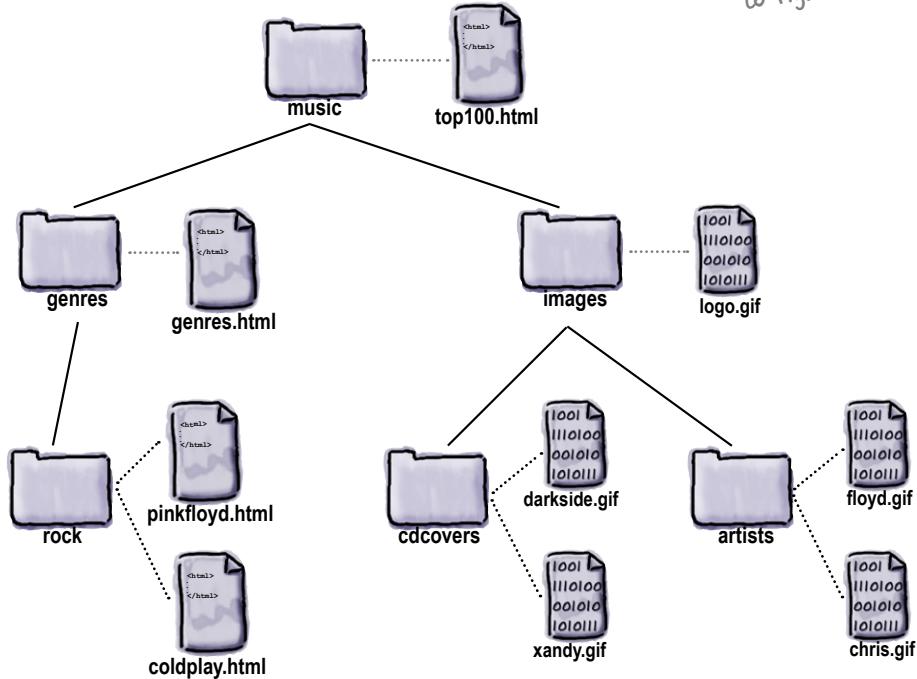
The Relativity Grand Challenge

Here's your chance to put your relativity skills to the test. We've got a website for the top 100 albums in a folder named "music". In this folder you'll find HTML files, other folders, and images. Your challenge is to find the relative paths we need so we can link from our web pages to other web pages and files.

On this page, you'll see the website structure; on the next page, you'll find the tasks to test your skills. For each source file and destination file, it's your job to make the correct relative path. If you succeed, you will truly be champion of relative paths.

Good luck!

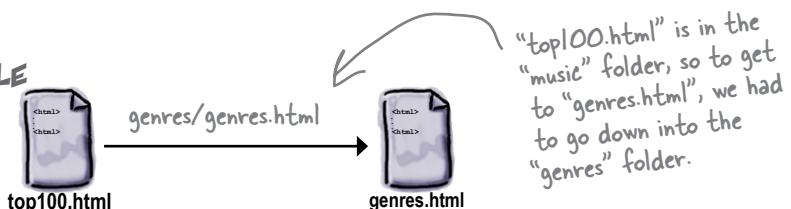
Feel free to draw right
on this website picture
to figure out the paths.



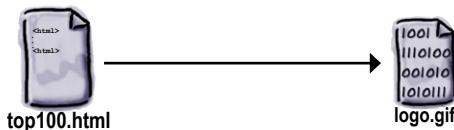
It's time for the competition to begin.

Ready...set...write!

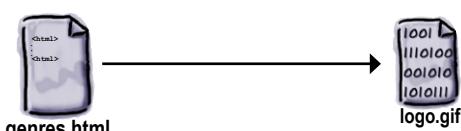
EXAMPLE



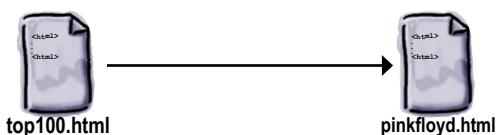
ROUND ONE



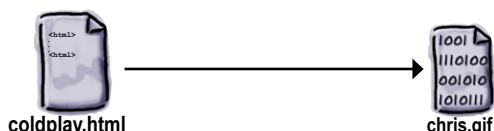
ROUND TWO



ROUND THREE



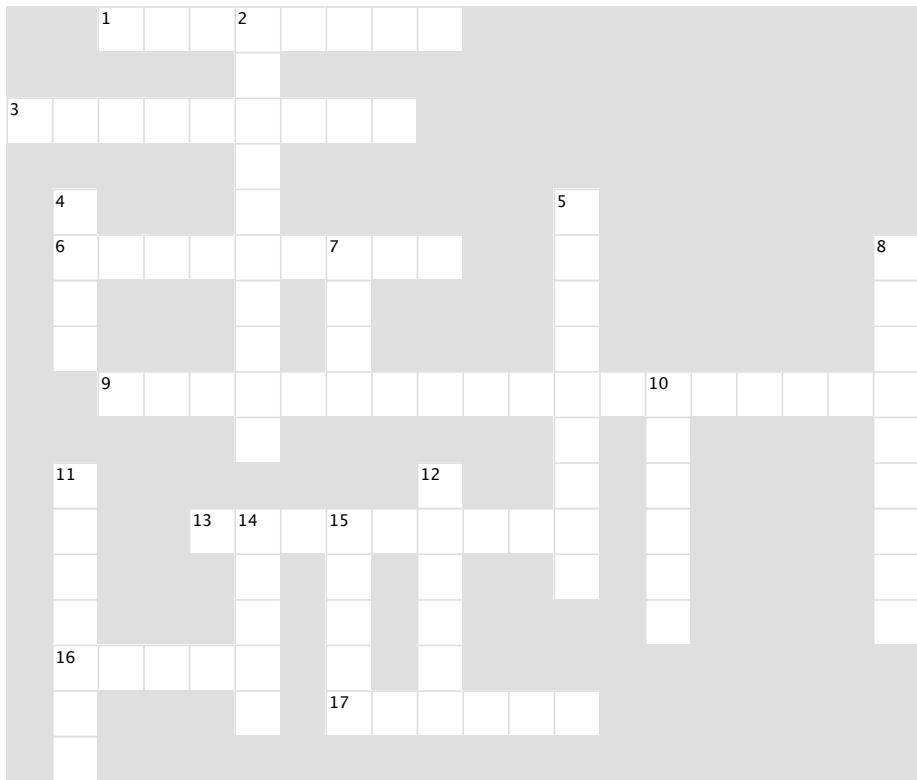
BONUS ROUND





HTMLcross

How does a crossword help you learn HTML? Well, all the words are HTML-related and from this chapter. In addition, the clues provide the mental twist and turns that will help you burn alternative routes to HTML right into your brain!



Across

1. “..*myfiles/index.html*” is this kind of link.
3. Another name for a folder.
6. Flavor of blue drink.
9. What href stands for.
13. Everything between the `<a>` and `` is this.
16. Can go in an `<a>` element, just like text.
17. Pronounced “..”.

Down

2. href and src are two of these.
4. Hardest-working attribute on the Web.
5. Rhymes with href.
7. Top folder of your site.
8. The “HT” in HTML.
10. Healthy drink.
11. A folder at the same level.
12. Use .. to reach this kind of directory.
14. Text between the `<a>` tags acts as a _____.
15. A subfolder is also called this.



Exercise SOLUTION

You needed to add a link with the label “Back to the Lounge” at the bottom of the elixir page that points back to “lounge.html”. Here’s our solution.

```

<html>
  <head>
    <title>Head First Lounge Elixirs</title>
  </head>
  <body>
    <h1>Our Elixirs</h1>

    <h2>Green Tea Cooler</h2>
    <p>
      
      Chock full of vitamins and mi
      combines the healthful benefi
      a twist of chamomile blossoms
    </p>
    <h2>Raspberry Ice Concentration</h2>
    <p>
      
      Combining raspberry juice wit
      citrus peel and rosehips, thi
      will make your mind feel clea
    </p>
    <h2>Blueberry Bliss Elixir</h2>
    <p>
      
      Blueberries and cherry essence mixed into a base
      of elderflower herb tea will put you in a relaxed
      state of bliss in no time.
    </p>
    <h2>Cranberry Antioxidant Blast</h2>
    <p>
      
      Wake up to the flavors of cranberry and hibiscus
      in this vitamin C rich elixir.
    </p>
    <p>
      <a href="lounge.html">Back to the Lounge</a>
    </p>
  </body>
</html>

```

Head First Lounge Elixirs
file:///chapter2/lounge/elixir.html

Our Elixirs

Green Tea Cooler



Chock full of vitamins and minerals, this elixir combines the healthful benefits of green tea with a twist of chamomile blossoms and ginger root.

Raspberry Ice Concentration



Combining raspberry juice with lemon grass, citrus peel and rosehips, this icy drink will make your mind feel clear and crisp.

Blueberry Bliss Elixir



Blueberries and cherry essence mixed into a base of elderflower herb tea will put you in a relaxed state of bliss in no time.

Cranberry Antioxidant Blast



Wake up to the flavors of cranberry and hibiscus in this vitamin C rich elixir.

[Back to the Lounge](#)

We put the link inside its own paragraph to keep things tidy. We'll talk more about this in the next chapter.

Here's the new `<a>` element pointing back to the lounge.

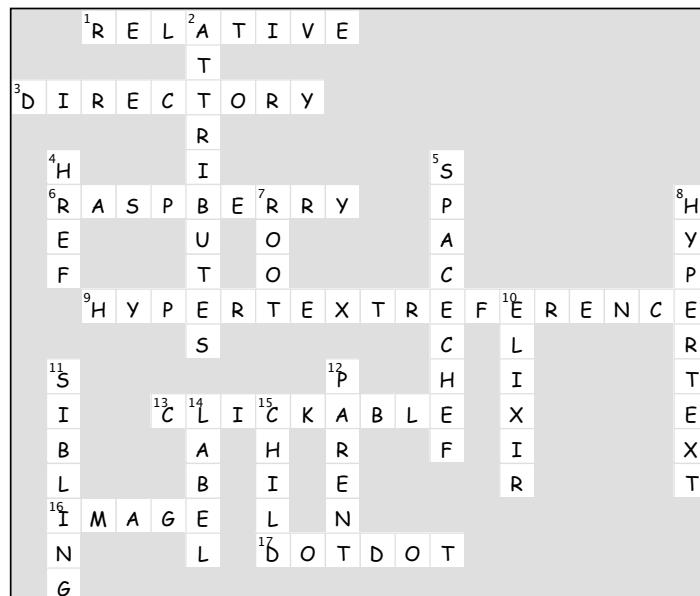




Exercise solutions



Label	Destination	Element
Hot or Not?	hot.html	Hot or Not?
Resume	cv.html	Resume
Eye Candy	candy.html	Eye Candy
See my mini	mini-cooper.html	See my mini
let's play	millionaire.html	 let's play

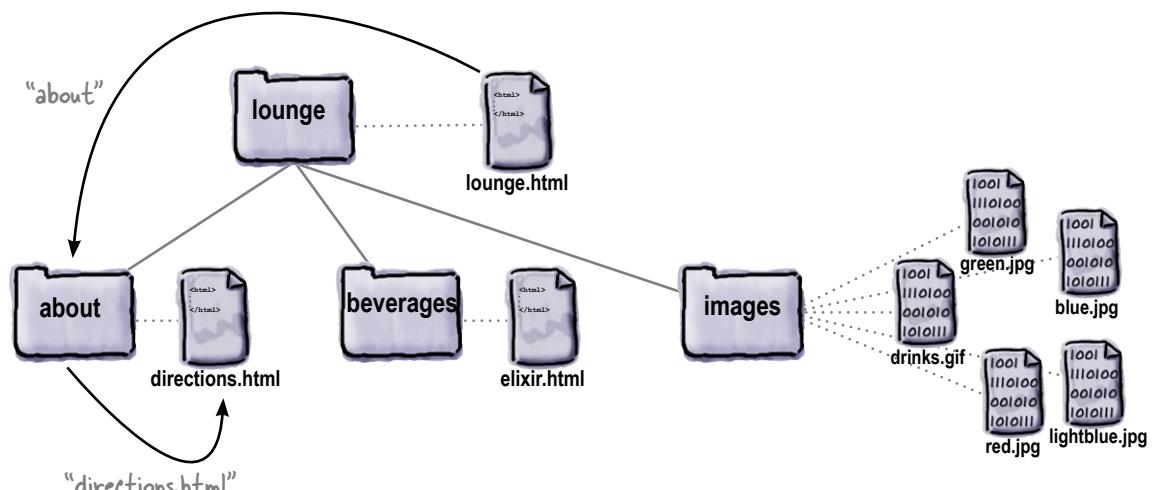




Sharpen your pencil Solution

Trace the relative path from "lounge.html" to "directions.html". When you've discovered it, complete the `<a>` element below.

Here's the solution. Did you change both `<a>` elements in "lounge.html"?



`detailed directions`

YOUR ANSWER HERE ↗



The Relativity Grand Challenge Solution

ROUND ONE



`top100.html` is in the music folder, so to get to `logo.gif`, we had to go down into the images folder.

ROUND TWO



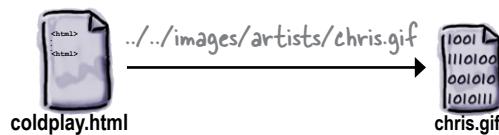
`genres.html` is down in the genres directory, so to get to `logo.gif`, we first had to go up to music, and then down into the images folder.

ROUND THREE



From `top100.html`, we go down into genres, then down into rock, and find `pinkfloyd.html`.

BONUS ROUND



This was a tricky one. From `coldplay.html`, which is down in the rock folder, we had to go up TWO folders to get to music, and then go down into images, and finally artists, to find the image `chris.gif`. Whew!

3 building blocks

Web Page Construction

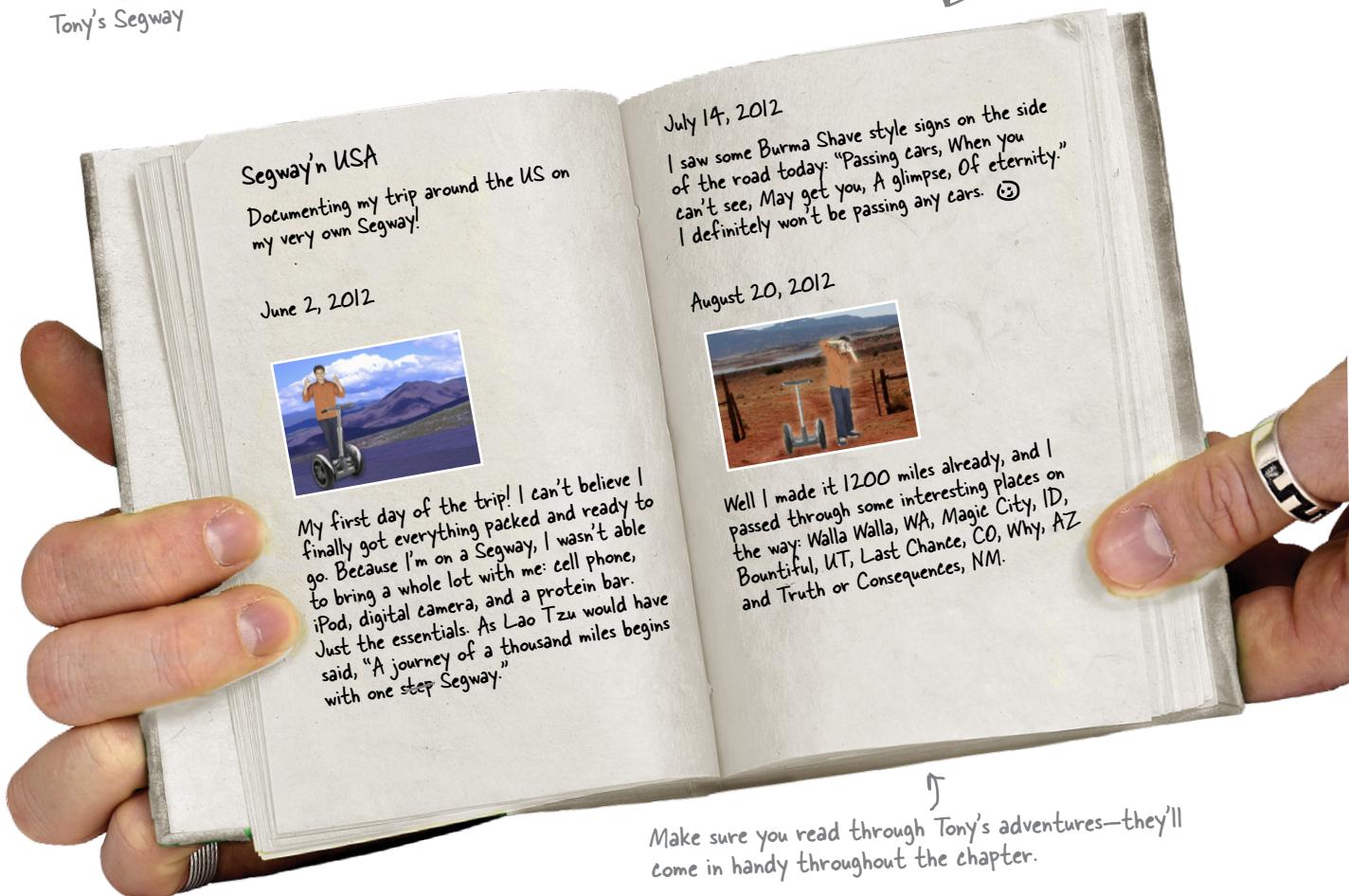
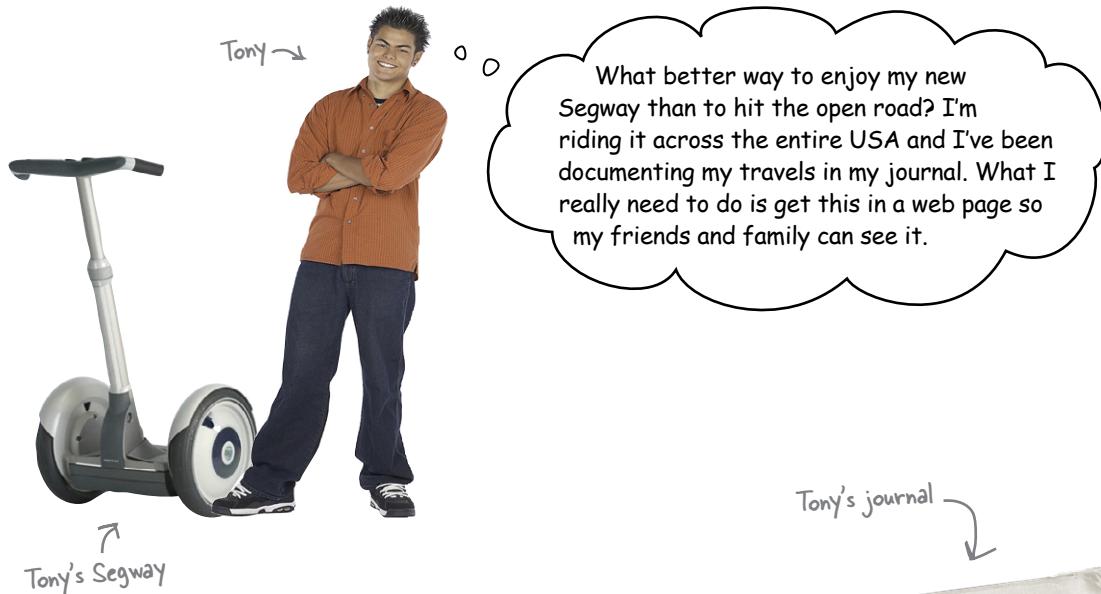
We better find some hard hats, Betty. It's a real construction zone around here, and these web pages are going up fast!



I was told I'd actually be creating web pages in this book?

You've certainly learned a lot already: tags, elements, links, paths...but it's all for nothing if you don't create some killer web pages with that knowledge. In this chapter we're going to ramp up construction: you're going to take a web page from conception to blueprint, pour the foundation, build it, and even put on some finishing touches. All you need is your hard hat and your toolbelt, as we'll be adding some new tools and giving you some insider knowledge that would make Tim "The Toolman" Taylor proud.

meet tony and his segway



From journal to website, at 12 mph

← recommended
The Segway's top speed.

Tony's got his hands full driving across the United States on his Segway.
Why don't you give him a hand and create a web page for him?

Here's what you're going to do:

- 1 First, you're going to create a rough sketch of the journal that is the basis for your page design.
- 2 Next, you'll use the basic building blocks of HTML (<h1>, <h2>, <h3>, <p>, and so on) to translate your sketch into an outline (or blueprint) for the HTML page.
- 3 Once you have the outline, then you're going to translate it into real HTML.
- 4 Finally, with the basic page done, you'll add some enhancements and meet some new HTML elements along the way.



Sharpen your pencil

Take a close look at Tony's journal and think about how you'd present the same information in a web page.

Draw a picture of that page on the right. No need to get too fancy; you're just creating a rough sketch. Assume all his journal entries will be on one page.

Things to think about:

- Think of the page in terms of large structural elements: headings, paragraphs, images, and so on.
- Are there ways his journal might be changed to be more appropriate for the Web?

Your sketch goes here.



STOP! Do this exercise before turning the page.

The rough design sketch

Tony's journal looks a lot like a web page; all we need to do to create the design sketch is to get all his entries on one page and map out the general organization. It looks like, for each day that Tony creates an entry, he has a date heading, an optional picture, and a description of what happened that day. Let's look at the sketch...

Tony gave his journal a title, "Segway'n USA," so let's get that right at the top as a heading.

He also gave his journal a description. We'll capture that here as a small paragraph at the top.

Each day, Tony creates an entry that includes the date, usually a picture, and a description of the day's adventures. So, that's a heading, an image, and another paragraph of text.

Sometimes he doesn't include a picture. In this entry, he just has a heading (the date) and a description of the day's events.

The third entry should look just like the first one: a heading, an image, and a paragraph.

Unlike Tony's paper journal, our page length isn't limited, so we can fit many entries on one web page. His friends and family can just use the scroll bar to scroll through his entries...

However, notice that we reversed the order of the journal entries from newest to oldest. That way, the most recent entries appear at the top where users can see them without scrolling.

Segway'n USA

Documenting my trip around the US on my very own Segway!

August 20, 2012



Well I made it 1200 miles already, and I passed through some interesting places on the way: Walla Walla, WA, Magic City, ID, Bountiful, UT, Last Chance, CO, Why, AZ and Truth or Consequences, NM.

July 14, 2012

I saw some Burma Shave style signs on the side of the road today: "Passing cars, When you can't see, May get you, A glimpse, Of eternity." I definitely won't be passing any cars.

June 2, 2012



My first day of the trip! I can't believe finally got everything packed and ready to go. Because I'm on a Segway, I wasn't able to bring a whole lot with me: cell phone, iPod, digital camera, and a protein bar. Just the essentials. As Lao Tzu would have said, "A journey of a thousand miles begins with one Segway."

From a sketch to an outline

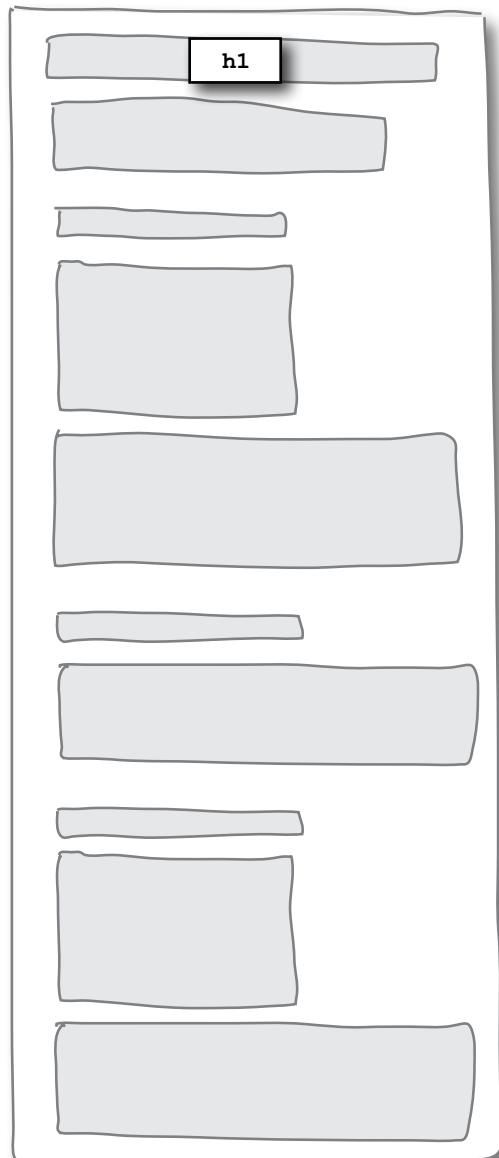
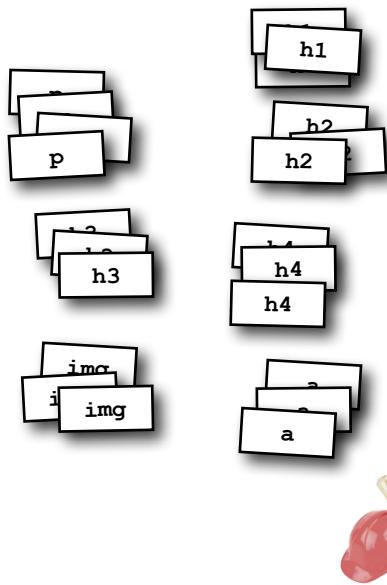
Now that you've got a sketch of the page, you can take each section and draw something that looks more like an outline or blueprint for the HTML page...

Here we've taken each area of the sketch and created a corresponding block in our blueprint.

All you need to do now is figure out which HTML element maps to each content area, and then you can start writing the HTML.

EXERCISE: WEB CONSTRUCTION

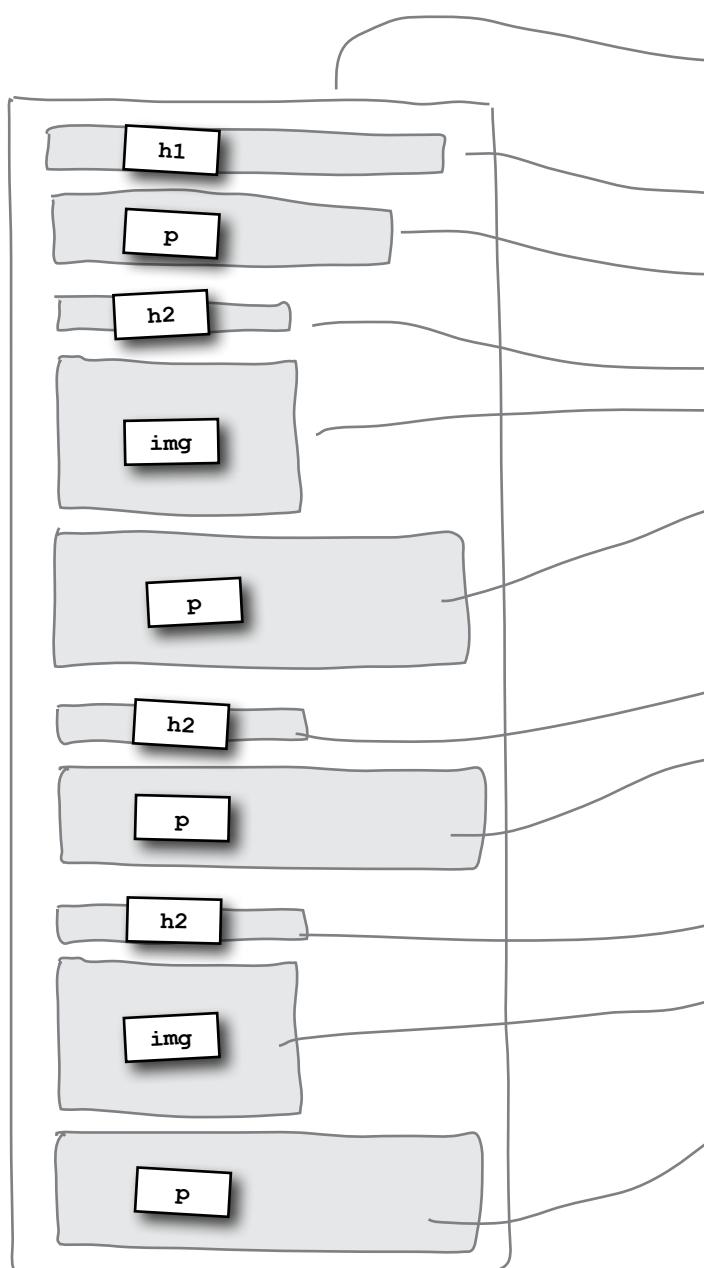
You've already figured out the major architectural areas of the page; now you just need to nail down the building materials. Use the elements below to label each area. You won't use them all, so don't worry if you have some building materials left over. And don't forget to wear your hard hat.



From the outline to a web page

You're almost there. You've created an outline of Tony's web page. Now all you need to do is create the corresponding HTML to represent the page and fill in Tony's text.

Before you begin, remember that every web page needs to start with the `<html>` element and include the `<head>` and `<body>` elements.



Don't forget, you always need the `<html>`, `<head>`, `<title>`, and `<body>` elements.

```
<html>
  <head>
    <title>My Trip Around the USA on a Segway</title>
  </head>
  <body>
```

We're using the title of the journal as the title of the web page.

```
    <h1>Segway' n USA</h1>
    <p>
```

Documenting my trip around the US on my very own Segway!

```
</p>
```

```
<h2>August 20, 2012</h2>
```

```

<p>
```

Well I made it 1200 miles already, and I passed through some interesting places on the way: Walla Walla, WA, Magic City, ID, Bountiful, UT, Last Chance, CO, Why, AZ and Truth or Consequences, NM.

```
</p>
```

```
<h2>July 14, 2012</h2>
```

```
<p>
```

I saw some Burma Shave style signs on the side of the road today: "Passing cars, When you can't see, May get you, A glimpse, Of eternity." I definitely won't be passing any cars.

```
</p>
```

```
<h2>June 2, 2012</h2>
```

```

<p>
```

My first day of the trip! I can't believe I finally got everything packed and ready to go. Because I'm on a Segway, I wasn't able to bring a whole lot with me: cell phone, iPod, digital camera, and a protein bar. Just the essentials. As Lao Tzu would have said, "A journey of a thousand miles begins with one Segway."

```
</p>
```

```
</body>
```

```
</html>
```

Here's the heading and description of Tony's journal.

heading
image
description

Here's Tony's most recent entry.

Here's his second entry, which doesn't have an image.

And at the bottom, Tony's first entry, with the image "segway1.jpg".

Last, but not least, don't forget to close your `<body>` and `<html>` elements.

Go ahead and type this in. Save your file to the "chapter3/journal" folder as "journal.html". You'll find the images "segway1.jpg" and "segway2.jpg" already in the "images" folder. When you're done, give this page a test drive.

Test driving Tony's web page

My Trip Around the USA on a Segway

+ file:///chapter3/journal/journal.html Google

Segway'n USA

Documenting my trip around the US on my very own Segway!

August 20, 2012



Well I made it 1200 miles already, and I passed through some interesting places on the way: Walla Walla, WA, Magic City, ID, Bountiful, UT, Last Chance, CO, Why, AZ and Truth or Consequences, NM

July 14, 2012

I saw some Burma Shave style signs on the side of the road today: Passing cars, When you can't see, May get you, A glimpse, Of eternity. I definitely won't be passing any cars.

June 2, 2012



My first day of the trip! I can't believe I finally got everything packed and ready to go. Because I'm on a Segway, I wasn't able to bring a whole lot with me: cellphone, iPod, digital camera, and a protein bar. Just the essentials. As Lao Tzu would have said, "A journey of a thousand miles begins with one Segway."



Look how well this page has come together. You've put everything in Tony's journal into a readable and well-structured web page.



Fantastic! This looks great; I can't wait to add more entries to my page.

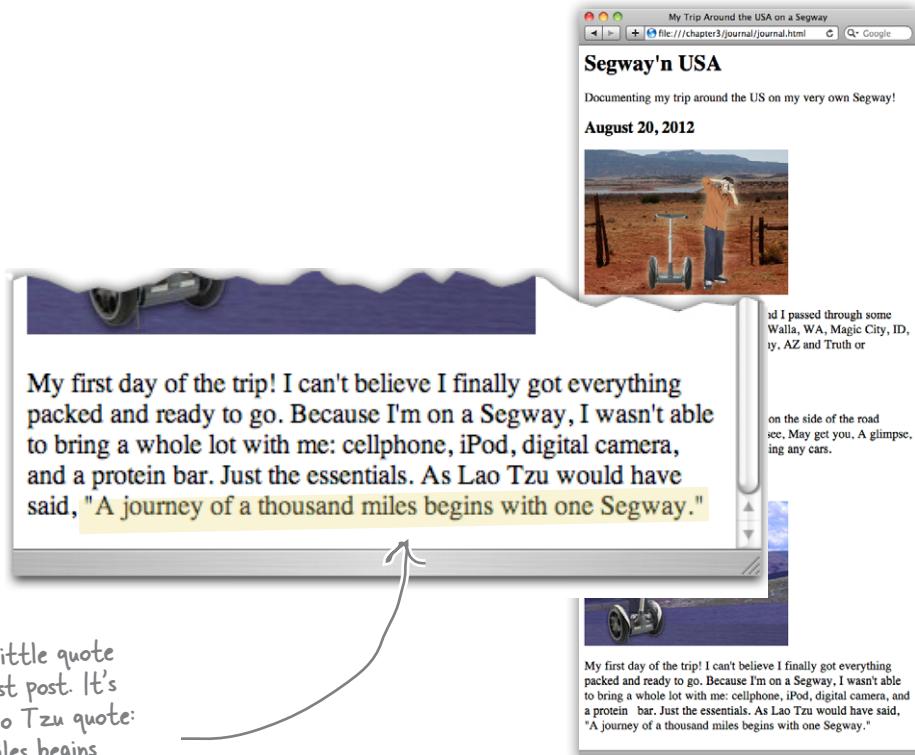


Tony's calling in from the road...

Adding some new elements

You have the basic elements of HTML down. You've gone from a hand-written journal to an online version in just a few steps using the basic HTML elements `<p>`, `<h1>`, `<h2>`, and ``.

Now we're going to s-t-r-e-t-c-h your brain a little and add a few more common elements. Let's take another look at Tony's journal and see where we can spruce things up a bit...



HTML has an element, `<q>`, for just that kind of thing. Let's take a look on the next page...

Meet the <q> element

Got a short quote in your HTML? The <q> element is just what you need. Here's a little test HTML to show you how it works:

```
<html>
  <head>
    <title>Quote Test Drive</title>
  </head>
  <body>
    <p>
      You never know when you'll need a good quote, how
      about <q>To be or not to be</q>, or <q>Wherever you go, there you are</q>.
    </p>
  </body>
</html>
```

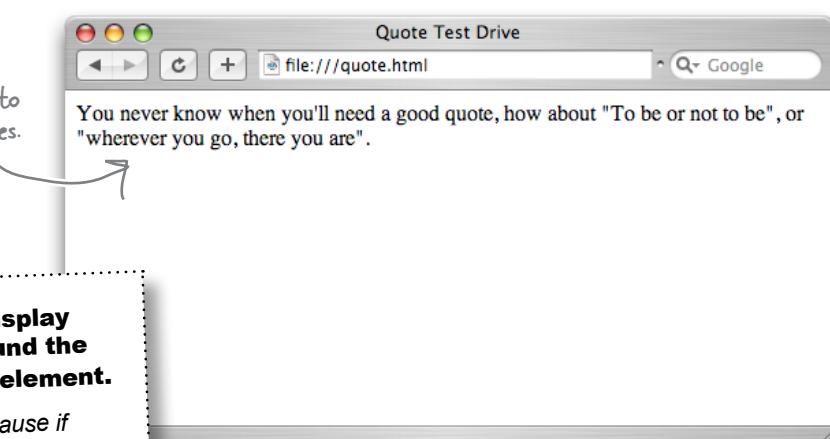
We've got two quotes in this HTML...

We surround each quote with a <q> opening tag and a </q> closing tag. Notice that we don't put our own double-quote characters around the quotes.

And test drive



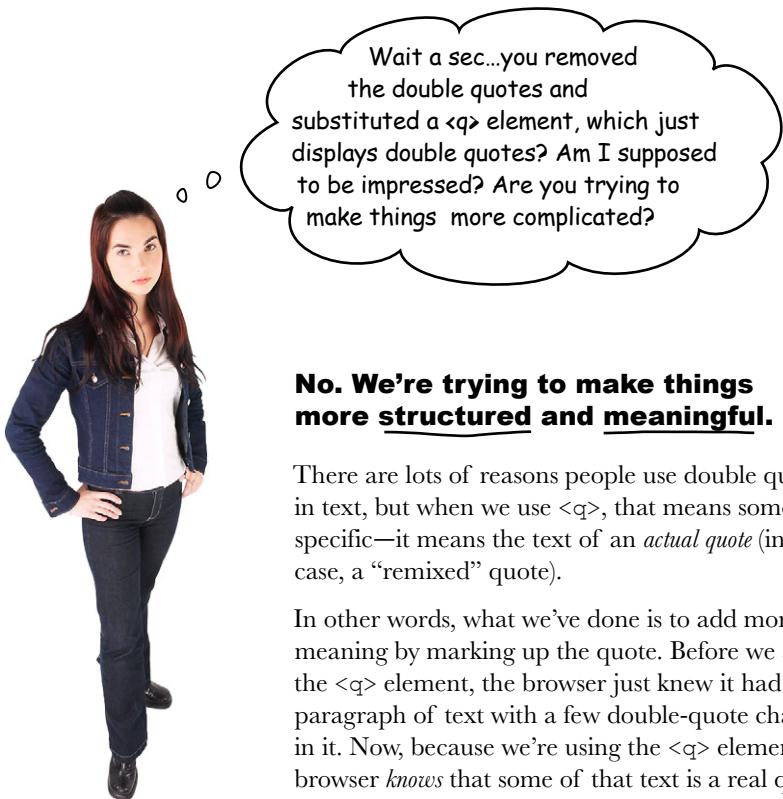
And here's how the quotes look in the browser. Notice the browser has gone to the trouble of adding the double quotes.



Watch it!

Not all browsers display double quotes around the content in the <q> element.

This is unfortunate, because if you add your own double quotes, some browsers will display TWO sets of quotes. We advise testing <q> in different browsers to see the results that you get.



No. We're trying to make things more structured and meaningful.

There are lots of reasons people use double quotes in text, but when we use `<q>`, that means something specific—it means the text of an *actual quote* (in Tony's case, a "remixed" quote).

In other words, what we've done is to add more meaning by marking up the quote. Before we added the `<q>` element, the browser just knew it had a paragraph of text with a few double-quote characters in it. Now, because we're using the `<q>` element, the browser *knows* that some of that text is a real quote.

See! Using double quotes doesn't make something an actual quote.

So what? Well, now that the browser knows this is a quote, it can display it in the best way possible. Some browsers will display double quotes around the text and some won't; and in instances where browsers are using non-English languages, other methods might be used. And don't forget mobile devices, like cell phones, or audio HTML browsers and screen readers for the visually impaired. It's also useful in other situations, such as a search engine that scours the Web looking for web pages with quotes. Structure and meaning in your pages are Good Things.

One of the best reasons (as you'll see when we get back to presentation and CSS later in the book) is that you'll be able to style quotes to look just the way you want. Suppose you want quoted text to be displayed in italics and colored gray? If you've used the `<q>` element to structure the quoted content in your web pages, you'll be able to do just that.



Here's Tony's journal. Go ahead and rework his Lao Tzu quote to use the <q> element. After you've done it on paper, make the changes in your "journal.html" file and give it a test drive. You'll find the solution in the back of the chapter.

```
<html>
  <head>
    <title>Segway'n USA</title>
  </head>
  <body>

    <h1>Segway'n USA</h1>
    <p>
      Documenting my trip around the US on my very own Segway!
    </p>

    <h2>August 20, 2012</h2>
    
    <p>
      Well I made it 1200 miles already, and I passed
      through some interesting places on the way: Walla Walla,
      WA, Magic City, ID, Bountiful, UT, Last Chance, CO,
      Why, AZ and Truth or Consequences, NM.
    </p>

    <h2>July 14, 2012</h2>
    <p>
      I saw some Burma Shave style signs on the side of the
      road today: "Passing cars, When you can't see, May get
      you, A glimpse, Of eternity." I definitely won't be passing
      any cars.
    </p>

    <h2>June 2, 2012</h2>
    
    <p>
      My first day of the trip! I can't believe I finally got
      everything packed and ready to go. Because I'm on a Segway,
      I wasn't able to bring a whole lot with me: cell phone, iPod,
      digital camera, and a protein bar. Just the essentials. As
      Lao Tzu would have said, "A journey of a thousand miles begins
      with one Segway."
    </p>
  </body>
</html>
```

Five-Minute Mystery



The Case of the Elements Separated at Birth

Identical twins were born in Webville a number of years ago, and by a freak accident involving an Internet router malfunction, the twins were separated shortly after birth. Both grew up without knowledge of the other, and only through another set of freak circumstances did they later meet and discover their identity, which they decided to keep secret.

After the discovery, they quickly learned that they shared a surprising number of things in common. Both were married to wives named Citation. They also both had a love for quotations. The first twin, the `<q>` element, loved short, pithy quotes, while the second, `<blockquote>`, loved longer quotes, often memorizing complete passages from books or poems.

Being identical twins, they bore a strong resemblance to each other, and so they decided to put together an evil scheme whereby they might stand in for each other now and then. They first tested this on their wives (the details of which we won't go into), and they passed with flying colors—their wives had no idea (or at least pretended not to).

Next they wanted to test their switching scheme in the workplace where, as another coincidence, they both performed the same job: marking up quotes in HTML documents. So, on the chosen day, the brothers went to the other's workplace fully confident they'd pull off their evil plan (after all, if their wives couldn't tell, how could their bosses?), and that's when things turned bad. Within 10 minutes of starting the work day, the brothers had both been found to be imposters and the standards authorities were immediately alerted.

***How were the twins caught in the act?
Keep reading for more clues...***

Loooooong quotes

Now that you know how to do short quotes, let's tackle long ones. Tony's given us a long quote with the Burma Shave jingle.

In his journal, Tony just put the Burma Shave quote right inside his paragraph, but wouldn't it be better if we pulled this quote out into a "block" of its own, like this:

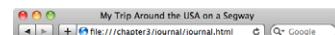
I saw some Burma Shave style signs on the side of the road today:

Passing cars,
When you can't see,
May get you,
A glimpse,
Of eternity.

If you don't know what
"Burma Shave" slogans are,
we'll tell you all about
them in just a few pages...

I definitely won't be passing any cars.

That's where the `<blockquote>` element comes in. Unlike the `<q>` element, which is meant for short quotes that are part of an existing paragraph, the `<blockquote>` element is meant for longer quotes that need to be displayed on their own.



Segway'n USA

Documenting my trip around the US on my very own Segway!

August 20, 2012

consequences, NM.

July 14, 2012

I saw some Burma Shave style signs on the side of the road today: "Passing cars, When you can't see, May get you, A glimpse, Of eternity." I definitely won't be passing any cars.

June 24, 2012

I saw some Burma Shave style signs on the side of the road today: Passing cars, When you can't see, May get you, A glimpse, Of eternity. I definitely won't be passing any cars.

June 2, 2012



My first day of the trip! I can't believe I finally got everything packed and ready to go. Because I'm on a Segway, I wasn't able to bring a whole lot with me: cellphone, iPod, digital camera, and a protein bar. Just the essentials. As Lao Tzu would have said, "A journey of a thousand miles begins with one Segway."

It's important to use the right tool for the job, and the `<blockquote>` element is perfect for this job.



Adding a <blockquote>

Let's get a <blockquote> into Tony's online journal.

- 1 Open your "journal.html" file and locate the July 14th entry. Rework the paragraph to look like this:

```
<h2>July 14, 2012</h2>
```

```
<p>
```

I saw some Burma Shave style signs on the side of the road today:

```
</p>
```

```
<blockquote>
```

Passing cars,
When you can't see,
May get you,
A glimpse,
Of eternity.

```
</blockquote>
```

```
<p>
```

I definitely won't be passing any cars.

```
</p>
```

To insert the <blockquote> element, we need to end this paragraph first.

Next we put the Burma Shave text in the <blockquote> element.

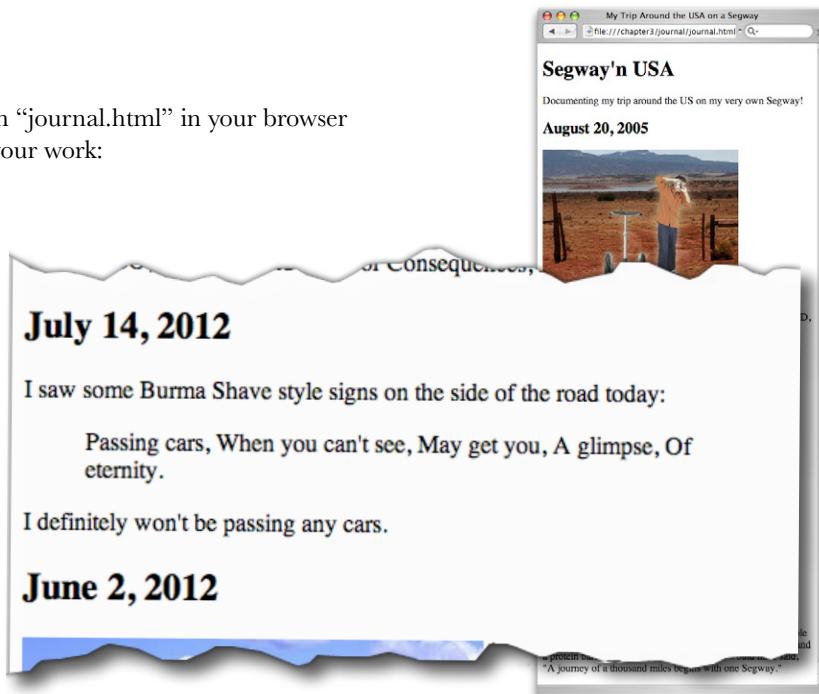
We also put each line of text on a separate line so it reads more like a Burma Shave slogan.

And finally, we need to add a <p> tag to start this paragraph after the <blockquote>.

- 2 Time for another test drive. Open "journal.html" in your browser and take a look at the results of your work:

<blockquote> creates a separate block (like <p> does), plus it indents the text a bit to make it look more like a quote. Just what we wanted...

But our quote isn't looking quite like we wanted because all the lines are running together. We really wanted them on different lines. Hmm. Let's come back to that in a bit...



^{there are no} Dumb Questions

Q: So let me see if I have this right: I use `<q>` when I just want to have some quote in with the rest of my paragraph, and I use `<blockquote>` when I have a quote that I want to break out on its own in my web page?

A: You've got it. In general you'll use `<blockquote>` if you want to quote something that was a paragraph or more, while you can use `<q>` anytime you just want to throw in a quote as part of your running text.

Q: Multiple paragraphs in a block quote? How do I do that?

A: Easy. Just put paragraph elements inside your `<blockquote>`, one for each paragraph. Do try this at home.

Q: How do I know what my quotes or block quotes will look like in other browsers? It sounds like they may handle it differently.

A: Yes. Welcome to the World Wide Web. You don't really know what your quotes will look like without trying them out in different browsers. Some browsers use double quotes, some use italics, and some use nothing at all. The only way to really determine how they'll look is to style them yourself, and we'll certainly be doing that later.

Q: I get that the `<blockquote>` element breaks its text out into a little block of its own and indents it, so why isn't the `<blockquote>` inside the paragraph, just like the `<q>` element is?

A: Because the `<blockquote>` really is like a new paragraph. Think about this as if you were typing it into a word processor. When you finish one paragraph, you hit the Return key twice and start a new paragraph. To type a block quote, you'd do the same thing and indent the quote. Put this in the back of your mind for a moment; it's an important point and we're going to come back to it in a sec.

Also, remember that the indenting is just the way some browsers display a `<blockquote>`. Some browsers might not use indentation for `<blockquote>`. So, don't rely on a `<blockquote>` to look the same in all browsers.

Q: Can I combine quote elements? For instance, could I use the `<q>` element inside the `<blockquote>` element?

A: Sure. Just like you can put a `<q>` element inside the `<p>` element, you can put `<q>` inside `<blockquote>`. You might do this if you're quoting someone who quoted someone else. But a `<blockquote>` inside a `<q>` doesn't really make sense, does it?

Q: You said that we can style these elements with CSS, so if I want to make the text in my `<q>` element italics and gray, I can do that with CSS. But couldn't I just use the `` element to italicize my quotes?

A: Well, you could, but it wouldn't be the right way to do it, because you'd be using the `` element for its effect on the display rather than because you're really writing emphasized text. If the person you were quoting really did emphasize a word, or you want to add emphasis to make a strong point about the quote, then go right ahead and use the `` element inside your quote. But don't do it simply for the italics. There are easier and better ways to get the look you want for your elements with CSS.

Solved: The Case of the Elements Separated at Birth

How were the identical quote twins found to be imposters so quickly?

As you've no doubt guessed by now, `<q>` and `<blockquote>` were discovered as soon as they went to work and began to mark up text. `<q>`'s normally unobtrusive little quotes were popping out into blocks of their own, while `<blockquote>`'s quotes were suddenly being lost inside regular paragraphs of text. In follow-up interviews with the victims of the pranks, one editor complained, "I lost an entire page of liner quotes thanks to these wackos." After being reprimanded and sent back to their respective jobs, `<blockquote>` and `<q>` fessed up to their wives, who immediately left town together in a T-Bird convertible. But that's a whole 'nother story (it didn't end well).



The real truth behind the `<q>` and `<blockquote>` mystery

Okay, it's time to stop the charade: `<blockquote>` and `<q>` are actually different types of elements. The `<blockquote>` element is a *block* element and the `<q>` element is an *inline* element. What's the difference? Block elements are always displayed as if they have a linebreak before and after them, while inline elements appear "in line" within the flow of the text in your page.



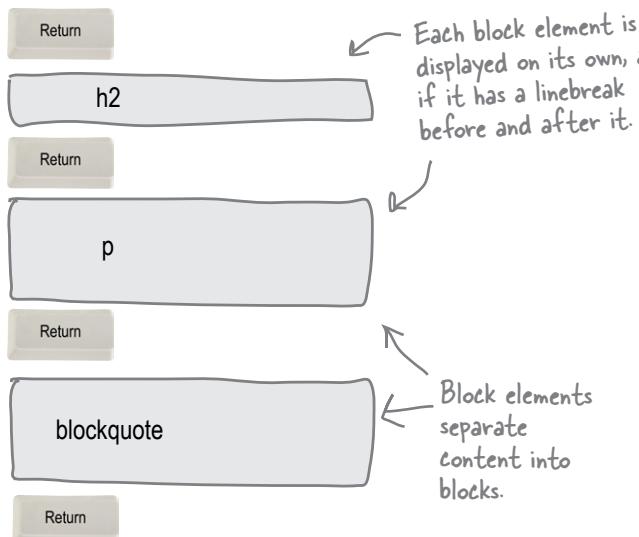
Block: stands on its own

`<h1>`, `<h2>`, ... , `<h6>`, `<p>`, and `<blockquote>` are all block elements.

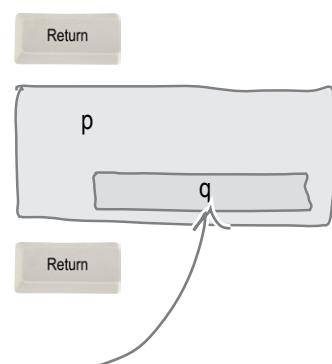


Inline: goes with the flow

`<q>`, `<a>`, and `` are inline elements.



`<q>`, on the other hand, like all inline elements, is just displayed in the flow of the paragraph it's in.



Remember: block elements stand on their own; inline elements go with the flow.

there are no
Dumb Questions



Q: I think I know what a linebreak is; it's like hitting the carriage return on a typewriter or the Return key on a computer keyboard. Right?

A: Pretty much. A linebreak is literally a "break in the line," like this, and happens when you hit the Return key, or on some computers, the Enter key. You already know that linebreaks in HTML files don't show up visually when the browser displays a page, right? But now you've also seen that anytime you use a block element, the browser uses linebreaks to separate each "block."

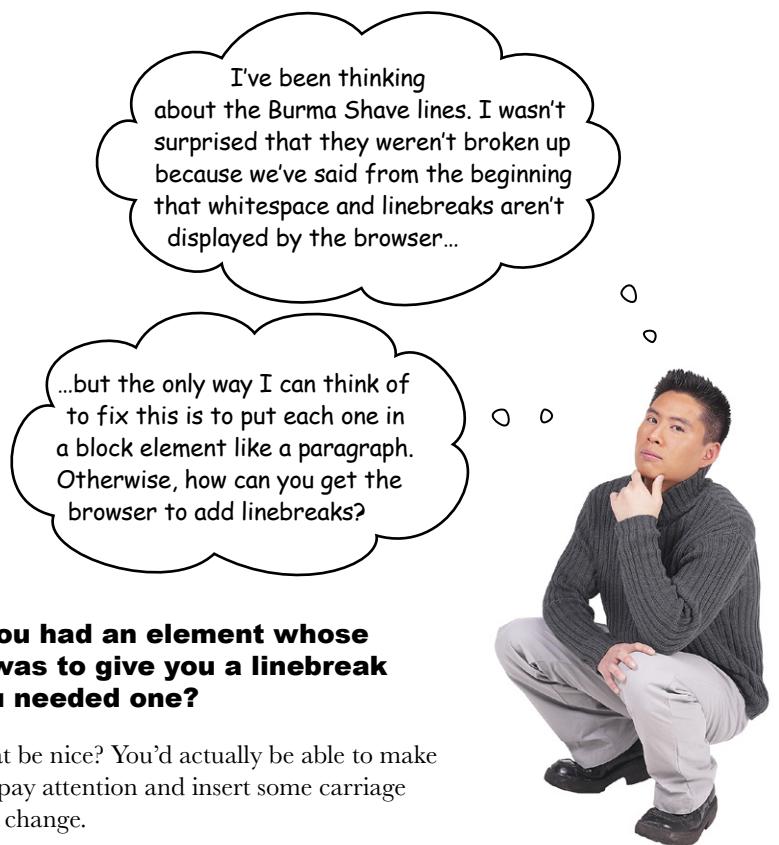


Once again, this all sounds great, but why is all this talk of linebreaks, blocks, and inline elements useful? Can we get back to web pages?

Don't underestimate the power of knowing how HTML works. You're soon going to see that the way you combine elements in a page has a lot to do with whether elements are displayed as block or inline. We'll get to all that.

In the meantime, you can also think about block versus inline this way: block elements are often used as the major building blocks of your web page, while inline elements usually mark up small pieces of content. When you're designing a page, you typically start with the bigger chunks (the block elements) and then add in the inline elements as you refine the page.

The real payoff is going to come when we get to controlling the presentation of HTML with CSS. If you know the difference between inline and block, you're going to be sipping martinis while everyone else is still trying to get their layout right.



What if you had an element whose only job was to give you a linebreak when you needed one?

Wouldn't that be nice? You'd actually be able to make the browser pay attention and insert some carriage returns for a change.

Turns out there is an element, the
 element, just for that purpose. Here's how you use it:

```
<h2>July 14, 2012</h2>
<p>
    I saw some Burma Shave style signs on the
    side of the road today:
</p>
<blockquote>
    Passing cars, <br>
    When you can't see, <br>
    May get you, <br>
    A glimpse, <br>
    Of eternity. <br>
</blockquote>
<p>
    I definitely won't be passing any cars.
</p>
```

Here's the July 14th snippet from Tony's page. ↗

Add a
 element to any line when you want to break the flow and insert a "linebreak."



Go ahead and add the
 elements to Tony's journal. After you make the changes, save the file, and give it a test drive.

Here's what the changes should look like. Now it reads like a Burma Shave slogan should read!



Chance, CO, Why, AZ and Truth or Consequences, NM.

July 14, 2012

I saw some Burma Shave style signs on the side of the road today:

Passing cars,
When you can't see,
May get you,
A glimpse,
Of eternity.



Each line now has a linebreak after it.

I definitely won't be passing any cars.

June 2, 2012



My first day of the trip! I can't believe I finally got everything packed and ready to go. Because I'm on a Segway, I wasn't able to bring a whole lot with me: cellphone, iPod, digital camera, and a protein bar. Just the essentials. As Lao Tzu would have said, "A journey of a thousand miles begins with one Segway."



In Chapter 1 we said that an element is an *opening tag + content + closing tag*. So how is `
` an element? It doesn't have any content, and it doesn't even have a closing tag.

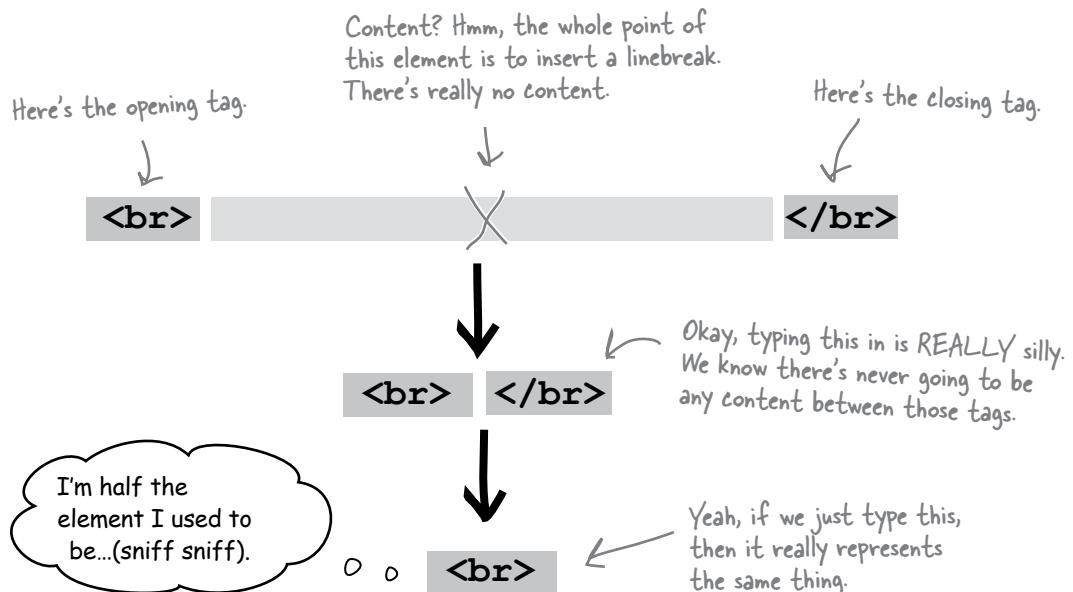
Exactly. It doesn't have any content.

The `
` element is an element that doesn't have any content. Why? Because it's just meant to be a linebreak, nothing else. So, when an element doesn't have any real content by design, we just use a shorthand to represent the element and it ends up looking like `
`. After all, if we didn't have this shorthand, you'd be writing `
</br>` every time you needed a linebreak, and how much sense does that make?

`
` isn't the only element like this; there are others, and we have a name for them: *void elements*. In fact, we've already seen another void element, the `` element. We'll be coming back to look at the `` element in detail in a couple chapters.

Keep in mind, the reason for the shorthand isn't laziness so much as it is efficiency. It's more efficient to represent void elements this way (efficient in typing, in the number of characters that end up in a page, and so on). In fact, after reading HTML for a while, you'll find that it is easier on your eyes too.

They used to be called "empty elements," which apparently made too much sense, so they renamed them to void. Personally, we still like empty.



there are no Dumb Questions

Q: So, the only purpose of
 is to insert a linebreak?

A: Right; the only place the browser typically inserts breaks in your content is when you start a new block element (like <p>, <h1>, and so on). If you want to insert a linebreak into your text, then you use the
 element.

Q: Why is
 called an “void” element?

A: Because it has no content, as in element = opening tag + content + closing tag. So, it's void because there's no content and no closing tag. Think like the “void of space”; there's nothing there, it's empty.

Q: I still don't get it. Explain why the
 element is “void”?

A: Think about an element like <h1> (or <p> or <a>). The whole point of the element is to mark up some content, like:

```
<h1>Don't wait, order now</h1>
```

With the
 element, the point is just to insert a linebreak into your HTML. There is no content you are trying to mark up. We don't need all the extra brackets and markup, so we just shorten it into a more convenient form. If you're thinking “void” is kind of a weird name, you're right: it comes from computer science and means “no value.”

Q: Are there any other void elements? I think must be a void element, too, right?

A: Yes, there are a couple of them. You've already seen us use the element, and we'll be getting to the details of this element soon.

Q: Can I make any element void? For instance, if I have a link, and don't want to give it any content, can I just write instead?

A: No. There are two types of elements in the world: normal elements, like <p>, <h1>, and <a>, and void elements, like
 and . You don't switch back and forth between the two. For instance, if you just typed , that's an opening tag without content or a closing tag (not good). If you write , that's an empty element and is perfectly fine, but isn't very useful in your page!

Q: I've seen pages not with
, but with
. What does that mean?

A: It means exactly the same thing. The syntax used in
 is a more strict syntax that works with XHTML. Whenever you see
, just think
, and unless you're planning on writing pages compliant with XHTML (see the appendix for more information on XHTML), you should just use
 in your HTML.

**Elements that don't have any content by design are called void elements. When you need to use a void element, like
 or , you only use an opening tag. This is a convenient shorthand that reduces the amount of markup in your HTML.**

Meanwhile, back at Tony's site...

You've come a long way already in this chapter: you've designed and created Tony's site, you've met a few new elements, and you've learned a few things about elements that most people creating pages on the Web don't even know (like block and inline elements, which are really going to come in handy in later chapters).

But you're not done yet. We can take Tony's site from good to great by looking for a few more opportunities to add some markup.

Like what? How about lists? Check this out:

There's a list right here. Tony wrote the list of cities that he's been through in his August journal entry.

The screenshot shows a web browser window titled "My Trip Around the USA on a Segway". The URL in the address bar is "file:///chapter3/journal/journal.html". The page content is a diary entry for July 1, 2012. The text reads: "Well I made it 1200 miles already, and I passed through some interesting places on the way: Walla Walla, WA, Magic City, ID, Bountiful, UT, Last Chance, CO, Why, AZ and Truth or Consequences, NM." A yellow callout box highlights the list of cities. Below the diary entry, there is another entry for July 14, 2012, which says: "I saw some Burma Shave style signs on the side of the road today: Passing cars, When you can't see, May get you, A glimpse, Of eternity." A blue callout box highlights the first sentence of this entry. At the bottom of the page, there is a photo of a man standing on a Segway with mountains in the background, and a caption: "My first day of the trip! I can't believe I finally got everything packed and ready to go. Because I'm on a Segway, I wasn't able to bring a whole lot with me: cellphone, iPod, digital camera, and a protein bar. Just the essentials. As Lao Tzu would have said, 'A journey of a thousand miles begins with one Segway.'"

Wouldn't it be great if we could mark up this text so the browser knows this text is a list? Then the browser could display the list items in a more useful way. Something like this:

Well I've made it 1200 miles already, and I passed through some interesting places on the way:

1. Walla Walla, WA
2. Magic City, ID
3. Bountiful, UT
4. Last Chance, CO
5. Why, AZ
6. Truth or Consequences, NM

Note that not only is this a list, but it's an ordered list. Tony visited these cities in a particular order.

Of course, you could use the `<p>` element to make a list...

It wouldn't be hard to make a list using the `<p>` element. It would end up looking something like this:

```
<p>
  1. Red Segway
</p>
<p>
  2. Blue Segway
</p>
```

Top two preferred colors for Segway.

But there are lots of reasons not to.

You should be sensing a common theme by now. You always want to choose the HTML element that is closest in meaning to the structure of your content. If this is a list, let's use a list element. Doing so gives the browser and you (as you'll see later in the book) the most power and flexibility to display the content in a useful manner.



Why not use `<p>` to make lists?
(Choose all that apply.)

- A. HTML has an element for lists. If you use that, then the browser knows the text is a list, and can display it in the best way possible.
- B. The paragraph element is really meant for paragraphs of text, not lists.
- C. It probably wouldn't look much like a list, just a bunch of numbered paragraphs.
- D. If you wanted to change the order of the list, or insert a new item, you'd have to renumber them all. That would suck.

Answer: A, B, C, & D

Constructing HTML lists in two easy steps

Creating an HTML list requires two elements that, when used together, form the list. The first element is used to mark up each *list item*. The second determines what kind of list you're creating: *ordered* or *unordered*.

Let's step through creating Tony's list of cities in HTML.

Step one:

Put each list item in an `` element.

To create a list, you put each list item in its own `` element, which means enclosing the content in an opening `` tag and a closing `` tag. As with any other HTML element, the content between the tags can be as short or as long as you like and broken over multiple lines.

We're just showing a fragment of the
HTML from Tony's journal here.

Locate this HTML in your "journal.html" file and
keep up with the changes as we make them.

```
<h2>August 20, 2012</h2>

<p>
```

Well I've made it 1200 miles already, and I passed
through some interesting places on the way:

First, move the list items outside of the paragraph. The
list is going to stand on its own.

```
<li>Walla Walla, WA</li>
<li>Magic City, ID</li>
<li>Bountiful, UT</li>
<li>Last Chance, CO</li>
<li>Why, AZ</li>
<li>Truth or Consequences, NM</li>
```

...and then enclose each list item
with an ``, `` set of tags.

Each of these ``
elements will become
an item in the list.

```
<h2>July 14, 2012</h2>
<p>
```

I saw some Burma Shave style signs on the side of
the road today:

```
</p>
```

Step two:

Enclose your list items with either the `` or `` element.

If you use an `` element to enclose your list items, then the items will be displayed as an ordered list; if you use ``, the list will be displayed as an unordered list. Here's how you enclose your items in an `` element.

Again, we're just showing a fragment of the HTML from Tony's journal here.

```
<h2>August 20, 2012</h2>

<p>
Well I've made it 1200 miles already, and I passed
through some interesting places on the way:
</p>
<ol>
<li>Walla Walla, WA</li>
<li>Magic City, ID</li>
<li>Bountiful, UT</li>
<li>Last Chance, CO</li>
<li>Why, AZ</li>
<li>Truth or Consequences, NM</li>
</ol>
<h2>July 14, 2012</h2>
<p>
I saw some Burma Shave style signs on the side of
the road today:
</p>
```

We want this to be an ordered list, because Tony visited the cities in a specific order. So we use an `` opening tag.

All the list items sit in the middle of the `` element and become its content.



Is `` a block element or inline? What about ``?

Make It Stick



unordered list = `ul`

ordered list = `ol`

list item = `li`



Taking a test drive through the cities

My Trip Around the USA on a Segway
file:///chapter3/journal/journal.html

Segway'n USA

Documenting my trip around the US on my very own Segway!

August 20, 2012

Well I made it 1200 miles already, and I passed through some interesting places on the way:

Well I m interestin

- 1. Wa
- 2. Ma
- 3. Bo
- 4. Las
- 5. Wh
- 6. Tru

July 1

I saw som today:

Pas
When you can't see,
May get you,
A glimpse,
Of eternity.

I definitely won't be passing any cars.

June 2, 2012

My first day of the trip! I can't believe I finally got everything packed and ready to go. Because I'm on a Segway, I wasn't able to bring a whole lot with me: cellphone, iPod, digital camera, and a protein bar. Just the essentials. As Lao Tzu would have said, "A journey of a thousand miles begins with one Segway."

Make sure you've added all the HTML for the list, reload your "journal.html" file and you should see something like this:

Here's the new and improved list of cities.

There's a linebreak before the list starts, so `` must be a block element.

But there's also a linebreak after each item, so `` must be a block element too!

Notice that the browser takes care of automatically numbering each list item (so you don't have to).



Sharpen your pencil

It turns out Tony actually visited Arizona after New Mexico. Can you rework the list so the numbering is correct?



Here's another list from Tony's journal: cell phone, iPod, digital camera, and a protein bar. You'll find it in his June 2nd entry. This is an *unordered* list of items.

The HTML for this entry is typed below. Go ahead and add the HTML to change the items into an HTML unordered list (remember, you use `` for unordered lists). We've already reformatted some of the text for you.

When you've finished, check your answers in the back of the chapter. Then make these changes in your "journal.html" file and test.

```
<h2>June 2, 2012</h2>

<p>
```

My first day of the trip! I can't believe I finally got everything packed and ready to go. Because I'm on a Segway, I wasn't able to bring a whole lot with me:

```
    cell phone
    iPod
    digital camera
    and a protein bar
```

Just the essentials. As Lao Tzu would have said, `<q>`A journey of a thousand miles begins with one Segway.`</q>`

```
</p>
```

there are no Dumb Questions

Q: Do I always have to use `` and `` together?

A: Yes, you should always use `` and `` together (or `` and ``). Neither one of these elements really makes sense without the other. Remember, a list is really a group of items: the `` element is used to identify each item, and the `` element is used to group them together.

Q: Can I put text or other elements inside an `` or `` element?

A: No, the `` and `` elements are designed to work only with the `` element.

Q: What about unordered lists? Can I make the bullet look different?

A: Yes. But hold that thought. We'll come back to that when we're talking about CSS and presentation.

Q: What if I wanted to put a list inside a list? Can I do that?

A: Yes, you sure can. Make the content of any `` either `` or ``, and you'll have a list within a list (what we call a *nested list*).

```
<ol>
  <li>Charge Segway</li>  Nested list
  <li>Pack for trip
    <ul>
      <li>cell phone</li>
      <li>iPod</li>
      <li>digital camera</li>
      <li>a protein bar</li>
    </ul>
  </li>
  <li>Call mom</li>
</ol>
```

Here's the ``. It encloses the nested list.

Q: I think I basically understand how block elements and inline elements are displayed by the browser, but I'm totally confused about what elements can go inside other elements, or, as you say, what can be "nested" inside of what.

A: That's one of the hardest things to get straight with HTML. This is something you're going to be learning for a few chapters, and we'll show you a few ways to make sure you can keep the relationships straight. But we're going to back up and talk about nesting a little more first. In fact, since you brought it up, we'll do that next.

Q: So HTML has ordered and unordered lists. Are there any other list types?

A: Actually, there is another type: definition lists. A definition list looks like this:

```
<dl>
  <dt>Burma Shave Signs</dt>
  <dd>Road signs common in the U.S. in
the 1920s and 1930s advertising shaving
products.</dd>
  <dt>Route 66</dt>
  <dd>Most famous road in the U.S. highway
system.</dd>
</dl>
```

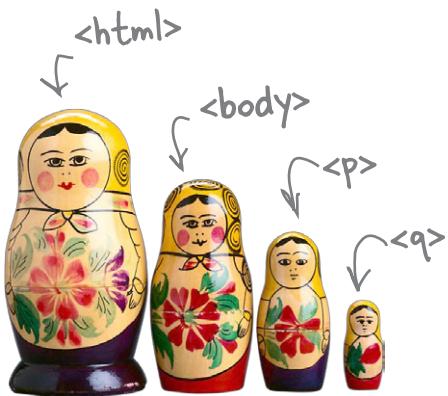
Each item in the list has a term, `<dt>`, and a description, `<dd>`.

Q: Burma Shave?

A: Burma Shave was a company that made brushless shaving cream in the early part of the 20th century. They began advertising their product using roadside signs in 1925, and these signs proved to be very popular (if somewhat distracting for drivers).

The signs were grouped in bunches of four, five, or six, each with one line from the slogan. At one point, there were 7,000 of these signs on roadsides throughout the United States. Most are gone now, but there are still a few left, here and there.

Type this in and give it a try.



Putting one element inside another is called “nesting”

When we put one element inside another element, we call that *nesting*. We say, “the `<p>` element is nested inside the `<body>` element.” At this point, you’ve already seen lots of elements nested inside other elements. You’ve put a `<body>` element inside an `<html>` element, a `<p>` element inside a `<body>` element, a `<q>` element inside a `<p>` element, and so on. You’ve also put a `<head>` element inside the `<html>` element, and a `<title>` element inside the `<head>`. That’s the way HTML pages get constructed.

The more you learn about HTML, the more important having this nesting in your brain becomes. But no worries—before long you’ll naturally think about elements this way.



To understand the nesting relationships, draw a picture

Drawing the nesting of elements in a web page is kind of like drawing a family tree. At the top you've got the great-grandparents, and then all their children and grandchildren below. Here's an example...

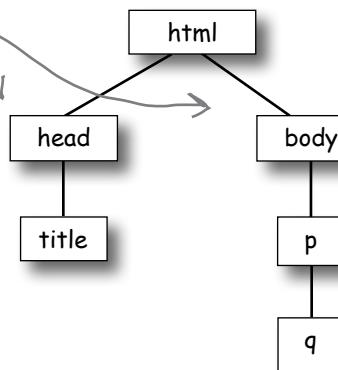
Simple web page

```
<html>
  <head>
    <title>Musings</title>
  </head>
  <body>
    <p>
      To quote Buckaroo,
      <q>The only reason
      for time is so
      that everything
      doesn't happen
      at once.</q>
    </p>
  </body>
</html>
```

`<html>` is always the element at the root of the tree.

`<html>` has two nested elements: `<head>` and `<body>`. You can call them both "children" of `<html>`.

`<title>` is nested within the `<head>` element.



Let's translate this into a diagram, where each element becomes a box, and each line connects the element to another element that is nested within it.

`<body>` is nested within the `<html>` element, so we say `<body>` is the "child" of `<html>`.

The parent of `<qp>` is `<p>`, the parent of `<p>` is `<body>`, and the parent of `<body>` is `<html>`.

Using nesting to make sure your tags match

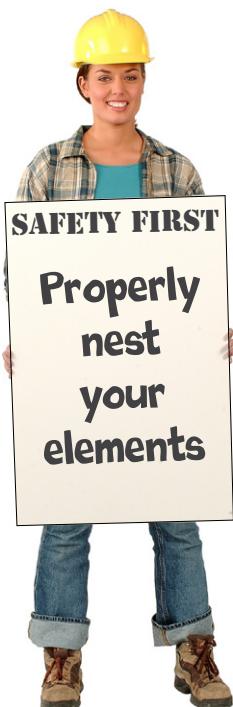
Your first payoff for understanding how elements are nested is that you can avoid mismatching your tags. (And there's gonna be more payoff later; just wait.)

What does “mismatching your tags” mean and how could that happen? Take a look at this example:

`<p>I'm so going to tweet this</p>`



Here's how this HTML looks;
`` is nested inside `<p>`.

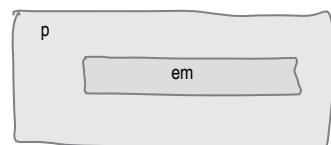


So far, so good, but it's also easy to get sloppy and write some HTML that looks more like this:

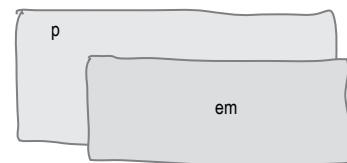
`<p>I'm so going to tweet this</p>`

WRONG: the `<p>` tag ends before the `` tag! The `` element is supposed to be inside the `<p>` element.

Given what you now know about nesting, you know the `` element needs to be nested fully within, or contained in, the `<p>` element.



GOOD: here the `` element is nested inside the `<p>`.



BAD: here the `` element has leaked outside of the `<p>` element, which means it's not properly nested inside it.

So what?

It's okay to mess up your nesting if you like playing Russian roulette. If you write HTML without properly nesting your elements, your pages may work on some browsers but not on others. By keeping nesting in mind, you can avoid mismatching your tags and be sure that your HTML will work in all browsers. This is going to become even more important as we get more into “industrial strength HTML” in later chapters.

BE the Browser

Below, you'll find an HTML file with some mismatched tags in it. Your job is to play like you're the browser and locate all the errors.

After you've done the exercise, look at the end of the chapter to see if you caught all the errors.



```
<html>
<head>
    <title>Top 100</title>
<body>
    <h1>Top 100
    <h2>Dark Side of the Moon</h2>
    <h3>Pink Floyd</h3>
    <p>
        There's no dark side of the moon; matter of fact <q>it's all dark.
    </p></q>
    <ul>
        <li>Speak to Me / Breathe</li>
        <li>On The Run</li>
        <li>Time</li>
        <li>The Great Gig in The Sky</li>
        <li>Money</li>
        <li>Us And Them</em>
        <li>Any Colour You Like</li>
        <li>Brain Damage</li>
        <li>Eclipse</li>
    </ul>
    </p>
    <h2>XandY</h2>
    <h3>Coldplay</h2>
    <ol>
        <li>Square One
        <li>What If?
        <li>White Shadows
        <li>Fix You
        <li>Talk
        <li>XandY
        <li>Speed of Sound
        <li>A Message
        <li>Low
        <li>Hardest Part
        <li>Swallowed In The Sea
        <li>Twisted Logic
    </ul>
    </body>
</head>
```



A bunch of HTML elements, in full costume, are playing a party game, “Who am I?” They’ll give you a clue—you try to guess who they are based on what they say. Assume they always tell the truth about themselves. Fill in the blanks to the right to identify the attendees. Also, for each attendee, write down whether or not the element is inline or block.

Tonight’s attendees:

Any of the charming HTML elements you’ve seen so far just might show up!

Name	Inline or block?
------	------------------

I'm the #1 heading.

.....

I'm all ready to link to another page.

.....

Emphasize text with me.

.....

I'm a list, but I don't have my affairs in order.

.....

I'm a real linebreaker.

.....

I'm an item that lives inside a list.

.....

I keep my list items in order.

.....

I'm all about image.

.....

Quote inside a paragraph with me.

.....

Use me to quote text that stands on its own.

.....



I was just creating a web page explaining everything I was learning from this book, and I wanted to mention the <html> element inside my page. Isn't that going to mess up the nesting? Do I need to put double quotes around it or something?

You're right, that can cause problems.

Because browsers use < and > to begin and end tags, using them in the content of your HTML can cause problems. But HTML gives you an easy way to specify these and other special characters using a simple abbreviation called a *character entity*. Here's how it works: for any character that is considered "special" or that you'd like to use in your web page, but that may not be a typeable character in your editor (like a copyright symbol), you just look up the abbreviation and then type it into your HTML. For example, the > character's abbreviation is >; and the < character's is <..

So, say you wanted to type "The <html> element rocks." in your page. Using the character entities, you'd type this instead:

The <html> element rocks.

Another important special character you should know about is the & (ampersand) character. If you'd like to have an & in your HTML content, use the character entity &; instead of the & character itself.

So what about the copyright symbol (that's ©right;)? And all those other symbols and foreign characters? You can look up common ones at this URL:

http://www.w3schools.com/tags/ref_entities.asp

or, for a more exhaustive list, use this URL:

<http://www.unicode.org/charts/>

there are no
Dumb Questions

Q: Wow, I never knew the browser could display so many different characters. There are a ton of different characters and languages at the www.unicode.org site.

A: Be careful. Your browser will only display all these characters if your computer or device has the appropriate fonts installed. So, while you can probably count on the basic entities from the www.w3schools.com page to be available on any browser, there is no guarantee that you can display all these entities. But, assuming you know something about your users, you should have a good idea of what kind of foreign language characters are going to be common on their machine.

Q: You said that & is special and I need to use the entity & in its place, but to type in any entity I have to use a &. So for, say, the > entity, do I need to type >?

A: No, no! The reason & is special is precisely because it is the first character of any entity. So, it's perfectly fine to use & in your entity names, just not by itself.

Just remember to use & anytime you type in an entity, and if you really need an & in your content, use & instead.

Q: When I looked up the entities at the www.w3schools.com, I noticed that each entity has a number too. What do I use that for?

A: You can use either the number, like d or the name of an entity in your HTML (they do the same thing). However, not all entities have names, so in those cases your only choice is to use the number.



Crack the Location Challenge

Dr. Evel, in his quest for world domination, has put up a private web page to be used by his evil henchmen. You've just received a snippet of intercepted HTML that may contain a clue to his whereabouts. Given your expert knowledge of HTML, you've been asked to crack the code and discover his location. Here's a bit of the text from his home page:

There's going to be an evil henchman meetup
next month at my underground lair in
Ðετröìτ. .
Come join us.

Hint: visit http://www.w3schools.com/tags/ref_entities.asp
and/or type in the HTML and see what your browser displays.

Element Soup

Whenever you want to make a link, you'll need the `<a>` element.

↙ `<a>`

Use this element for short quotes...you know, like "to be or not to be," or "No matter where you go, there you are."

↙ `<q>`

Just give me a paragraph, please.

The code element is used for displaying code from a computer program.

↖ `<code>`

``

Use this element to mark up text you'd say in a different voice, like if you are emphasizing a point.



``

Use this element to mark up text you want emphasized with extra strength.



`<pre>`

Use this element for formatted text when you want the browser to show your text exactly as you typed it.

`<time>`

This element tells the browser that the content is a date or time, or both.

``

Need to display a list? Say, a list of ingredients in a recipe or a to-do list? Use the `` element.

``

If you need an ordered list instead, use the `` element.

`` is for items in lists, like chocolate, hot chocolate, chocolate syrup...

``

This is for lengthy quotations—something that you want to highlight as a longer passage, say, from a book.

↖ `<blockquote>`

`
`

This is an element for including an image, like a photo, in your page.

``

Here are a bunch of elements you already know, and a few you don't.

Remember, half the fun of HTML is experimenting! So make some files of your own and try these out.

Rockin' page. It's perfect for my trip and it really does a good job of providing an online version of my journal. You've got the HTML well organized too, so I should be able to add new material myself. So, when can we actually get this off your computer and onto the Web?



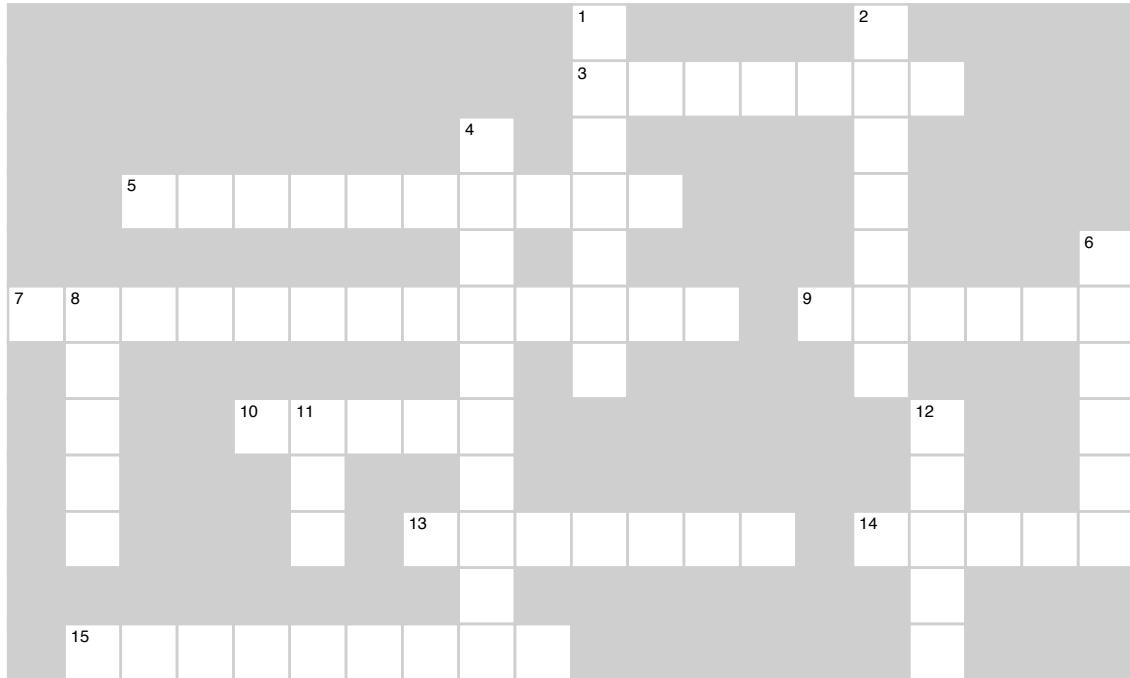
BULLET POINTS

- Plan the structure of your web pages before you start typing in the content. Start with a sketch, then create an outline, and finally write the HTML.
- Plan your page starting with the large, block elements, and then refine with inline elements.
- Remember, whenever possible, use elements to tell the browser what your content means.
- Always use the element that most closely matches the meaning of your content. For example, never use a paragraph when you need a list.
- <p>, <blockquote>, , , and are all block elements. They stand on their own and are displayed (by default) with a linebreak above and below the content within them.
- <q> and are inline elements. The content in these elements flows in line with the rest of the content in the containing element.
- Use the
 element when you need to insert your own linebreaks.
-
 is a “void” element.
- Void elements have no content.
- A void element consists of only one tag.
- An “empty” element has no content. But it does have both opening and closing tags.
- A nested element is an element contained completely within another element. If your elements are nested properly, all your tags will match correctly.
- You make an HTML list using two elements in combination: use with for an ordered list; use with for an unordered list.
- When the browser displays an ordered list, it creates the numbers for the list so you don't have to.
- You can build nested lists within lists by putting or elements inside your elements.
- Use character entities for special characters in your HTML content.



HTMLcross

It's time to give your right brain a break and put that left brain to work: all the words are HTML-related and from this chapter.



Across

1. Tony's transportation.
8. Famous catchy road signs.
10. <q> is this type of element.
11. Another void element.
13. Element without content.
14. Major building blocks of your pages.
15. Use for these kinds of lists.

Down

2. Left together in a T-bird.
3. Max speed of a Segway.
4. Tony won't be doing any of this.
5. Putting one element inside another is called this.
6. Requires two elements.
7. Block element for quotes.
9. Use for these kinds of lists.
12. Void elements have none.



Exercise Solution

Here's the rework of Tony's Lao Tzu quote using the `<q>` element. Did you give your solution a test drive?

Here's the part that changes...

`<p>`

My first day of the trip! I can't believe I finally got everything packed and ready to go. Because I'm on a Segway, I wasn't able to bring a whole lot with me: cell phone, iPod, digital camera, and a protein bar. Just the essentials. As Lao Tzu would have said, `<q>A journey of a thousand miles begins with one Segway.</q>`

`</p>`

We've added the `<q>` opening tag before the start of the quote and the `</q>` closing tag at the very end.

Notice that we also removed the double quotes.

And here's the test drive...

Okay, it doesn't LOOK any different, but don't you FEEL better now?

The screenshot shows a web browser window with a dark header bar. Below the header, there is a dark blue decorative bar featuring a car icon. The main content area contains the following text:

```

    My first day of the trip! I can't believe I finally got everything
    packed and ready to go. Because I'm on a Segway, I wasn't able
    to bring a whole lot with me: cellphone, iPod, digital camera,
    and a protein bar. Just the essentials. As Lao Tzu would have
    said, "A journey of a thousand miles begins with one Segway."
  
```



Here's another list from Tony's journal: cell phone, iPod, digital camera, and a protein bar. You'll find it in his June 2 entry. This is an *unordered* list of items.

Make these changes in your "journal.html" file, too. Does it look like you expected?

```
<h2>June 2, 2012</h2>

<p>
```

My first day of the trip! I can't believe I finally got everything packed and ready to go. Because I'm on a Segway, I wasn't able to bring a whole lot with me:

```
</p>
<ul>
    <li>cell phone</li>
    <li>iPod</li>
    <li>digital camera</li>
    <li>and a protein bar</li>
</ul>
<p>
```

Just the essentials. As Lao Tzu would have said, <q>A journey of a thousand miles begins with one Segway.</q>

First, end the previous paragraph.
Start the unordered list.
Put each item into an element.
End the unordered list.
And we need to start a new paragraph.



```

<html>
<head>
    <title>Top 100</title>
<body>
    Top 100
    <h2>Dark Side of the Moon</h2>
    <h3>Pink Floyd</h3>
    <p>
        There's no dark side of the moon; matter of fact <q>it's all dark.
    </p></q>
    <ul>
        <li>Speak to Me / Breathe</li>
        <li>On The Run</li>
        <li>Time</li>
        <li>The Great Gig in The Sky</li>
        <li>Money</li>
        <li>Us And Them</em>
        <li>Any Colour You Like</li>
        <li>Brain Damage</li>
        <li>Eclipse</li>
    </ul>
    </p>
    <h2>XandY</h2>
    <h3>Coldplay</h3>
    <ol>
        <li>Square One
        <li>What If?
        <li>White Shadows
        <li>Fix You
        <li>Talk
        <li>Xandy
        <li>Speed of Sound
        <li>A Message
        <li>Low
        <li>Hardest Part
        <li>Swallowed In The Sea
        <li>Twisted Logic
    </ol>
    </ul>
</body>
</head>

```

Missing </head> closing tag

Missing </h1> closing tag

<p> and <q> are not nested properly: the </p> tag should come after the </q> tag.

We have a closing where we should have a closing tag.

Here's a closing </p> that doesn't match any opening <p> tag.

We mixed up the closing </h2> and </h3> tags on these headings.

We started an list, but it's matched with a closing tag.

We're missing all our closing tags.

This doesn't match the opening tag at the start of the list above.

Here's our missing </head> tag, but we're missing a closing </html> tag.



Exercise SOLUTION

A bunch of HTML elements, in full costume, are playing a party game “Who am I?” They gave you a clue—you tried to guess who they were based on what they said.

Tonight’s attendees:

Quite a few of the charming HTML elements you’ve seen so far showed up for the party!

I’m the #1 heading.

I’m all ready to link to another page.

Emphasize text with me.

I’m a list, but I don’t have my affairs in order.

I’m a real linebreaker.

I’m an item that lives inside a list.

I keep my list items in order.

I’m all about image.

Quote inside a paragraph with me.

Use me to quote text that stands on its own.



Name

Inline or
block?

h1

a

em

ul

br

li

ol

img

q

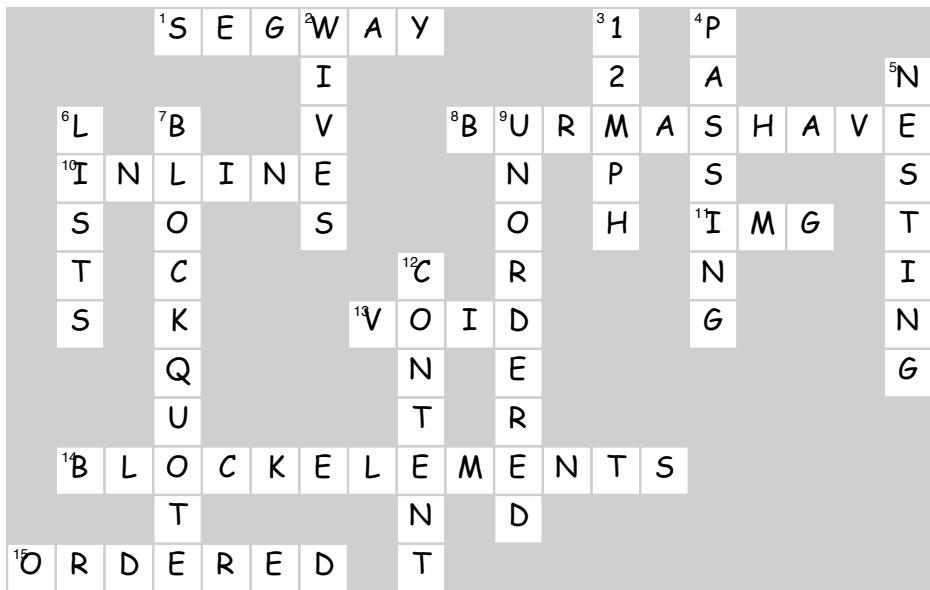
blockquote

Hmm, it looks like an inline, BUT `<a>` can wrap block elements, not just text. So, depending on the context, `<a>` can be either inline or block.

Stumped?
`
` is in limbo land between block and inline.

It does create a linebreak, but doesn’t break a bit of text into two separate blocks, like if you had two `<p>` elements.

We haven’t talked about this in detail yet, but, yes, `` is inline. Give it some thought and we’ll come back to this in Chapter 5.



Crack the Location Challenge

You could have looked up each entity or typed them in. In either case, the answer looks like Detroit!

There's going to be an evil henchman meetup next month at my underground lair in Ðετröìτ. Come join us.

There's going to be an evil henchman meetup next month at my underground lair in Detroit. Come join us.

4 getting connected

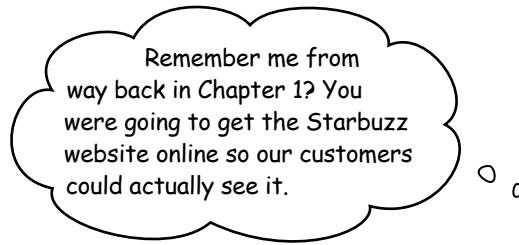
A Trip to Webville



Web pages are a dish best served on the Internet.

So far you've only created HTML pages that live on your own computer. You've also only linked to pages that are on your own computer. We're about to change all that. In this chapter we'll encourage you to get those web pages on the Internet where all your friends, fans, and customers can actually see them. We'll also reveal the mysteries of linking to other pages by cracking the code of the h, t, t, p, :, /, /, w, w, w. So, gather your belongings; our next stop is Webville.

WARNING: once you get to Webville, you may never come back.
Send us a postcard.



Getting Starbuzz (or yourself) onto the Web

You're closer to getting Starbuzz—or even better, your own site—on the Web than you might think. All you need to do is find a “web hosting company” (we'll call this a “hosting company” from now on) to host your pages on their servers, and then copy your pages from your computer to one of those servers.

Of course it helps to understand how your local folders are going to “map” to the server's folders, and once you put your pages on the server, how you point a browser to them. But we'll get to all that. For now, let's talk about getting you on the Web. Here's what you're going to need to do:

- ① Find yourself an hosting company.**
- ② Choose a name for your site (like www.starbuzzcoffee.com).**
- ③ Find a way to get your files from your computer to a server at the hosting company (there are a few ways).**
- ④ Point your friends, family, and fans to your new site and let the fun begin.**



We're going to take you through each of these steps, and even if you're not going to set up a website online *right now*, follow along because you'll learn some important things you'll need to know later. So, get ready for a quick detour from HTML....



Finding a hosting company

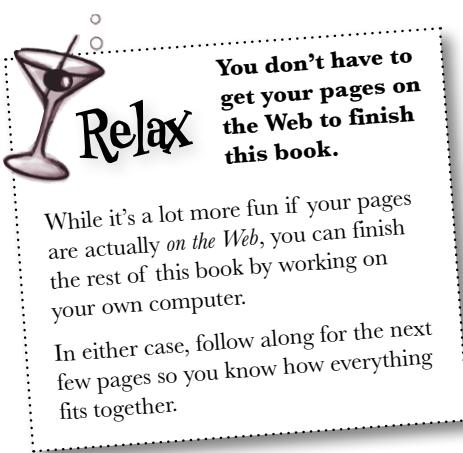
To get your pages on the Web, you need a server that actually lives on the Web *full-time*. Your best bet is to find a hosting company and let them worry about the details of keeping a server running. No worries, though; finding a hosting company is fairly straightforward and inexpensive.

Which company? Well, we'd *love* to sign you up for web hosting at **Head First Hip Web Hosting, Inc.**, but that doesn't really exist. So, you're going to have to do a little homework on your own. While finding a company to host your pages isn't difficult, it's kind of like choosing a cable TV company: there are lots of options and plans. You really have to shop around for the best deals and for the service that works for you.

The good news is that you should be able to get started for almost nothing out of your pocket, and you can always upgrade later if you need additional features. While we can't suggest a particular provider, we can tell you a few things to look for in a provider, and we also list a few of the more popular providers at:

<http://wickedlysmart.com/hosting-providers/>

Note from marketing:
if a hosting company
writes a big enough
check, we can!



One-minute hosting guide

We can't tell you everything you need to know about getting a hosting company (after all, this book is about HTML and CSS), but we're going to give you a good push in the right direction. Here are some features to think about while you're shopping.

- **Technical support:** Does the hosting company have a good system for handling your technical questions? The better ones will answer your questions quickly either over the phone or via email.
- **Data transfer:** This is a measure of the amount of pages and data the hosting company will let you send to your visitors during a given month. Most hosting companies offer reasonable amounts of data transfer for small sites in their most basic plans. If you're creating a site that you expect will have lots of visitors, you may want to carefully look into this.
- **Backups:** Does the hosting company regularly make a backup of your pages and data that can be recovered in the event that the server has a hardware failure?
- **Domain names:** Does the hosting company include a domain name in its pricing? More about these on the next page.
- **Reliability:** Most hosting companies report keeping websites up 99% of the time or better.
- **Goodies:** Does your package include other goodies such as email addresses, forums, or support for scripting languages (something that may become important to you in the future)?



domain HELLO, my^ name is...

Even if you've never heard of a *domain name*, you've seen and used a zillion of them; you know...google.com, facebook.com, amazon.com, disney.com, and maybe a few you wouldn't want us to mention.

So what is a domain name? Just a unique name that is used to locate your site. Here's an example:

This part is the domain name.
www.starbuzzcoffee.com

This part is the name of a specific server IN the domain.
↑
There are different domain "endings" for different purposes: .com, .org, .gov, .edu; and also for different countries: .co.uk, .co.jp, and so on. When choosing a domain, pick the one that best fits you.

There are a couple of reasons you should care about domain names. If you want a unique name for your site, you're going to need your own domain name. Domain names are also used to link your pages to other websites (we'll get to that in a few pages).

There is one other thing you should know. Domain names are controlled by a centralized authority (ICANN) to make sure that only one person at a time uses a domain name. Also (you knew it was coming), you pay a small annual registration fee to keep your domain name.

How can you get a domain name?

The easy answer is to let your hosting company worry about it. They'll often throw in your domain name registration with one of their package deals. However, there are hundreds of companies that would be glad to help—you can find a list of them at:

<http://www.internic.net/regist.html>

As with finding a hosting company, we're afraid we'll have to leave you to find and register your own domain name. You'll probably find that going through your hosting company is the easiest way to get that done.

After years of struggling, we finally have our very own domain name.





there are no Dumb Questions

Q: Why is it called a “domain name” rather than a “website name”?

A: Because they are different things. If you look at www.starbuzzcoffee.com, that's a website name, but only the “starbuzzcoffee.com” part is the domain name. You could also create other websites that use the same domain name, like corporate.starbuzzcoffee.com or employees.starbuzzcoffee.com. So the domain name is something you can use for a lot of websites.

Q: If I were going to get the domain name for Starbuzz, wouldn't I want to get the name www.starbuzzcoffee.com? Everyone seems to use websites with the www at the front.

A: Again, don't confuse a domain name with a website name: starbuzzcoffee.com is a domain name, while www.starbuzzcoffee.com is the name of a website. Buying a domain is like buying a piece of land; let's say, 100mainstreet.com. On that land, you can build as many websites as you like, for example: home.100mainstreet.com, toolshed.100mainstreet.com, and outhouse.100mainstreet.com. So www.starbuzzcoffee.com is just one website in the starbuzzcoffee.com domain.

Q: What's so great about a domain name anyway? Do I really need one? My hosting company says I can just use their name, www.dirtcheaphosting.com?

A: If that meets your needs, there is nothing wrong with using their name. But (and it's a big but) here's the disadvantage: should you ever want to choose another hosting company, or should that hosting company go out of business, then everyone who knows your site will no longer be able to easily find it. If, on the other hand, you have a domain name, you can just take that with you to your new hosting company (and your users will never even know you've switched).

Q: If domain names are unique, that means someone might already have mine. How can I find out?

A: Good question. Most companies that provide registration services for domain names allow you to search to see if a name is taken (kind of like searching for vanity license plates). You'll find a list of these companies at <http://www.internic.net/regist.html>.

Here's an exercise you really need to go off and do on your own. We'd love to personally help, but there's only so much you can ask of book authors (and feeding the cat while you're on vacation is probably out too).



DO try this at home

It's time to seek out a hosting company and grab a domain name for your site. Remember, you can visit Wickedly Smart for some suggestions and resources. Also remember that you can complete the book without doing this (even though you really should!).

My Web Hosting Company:

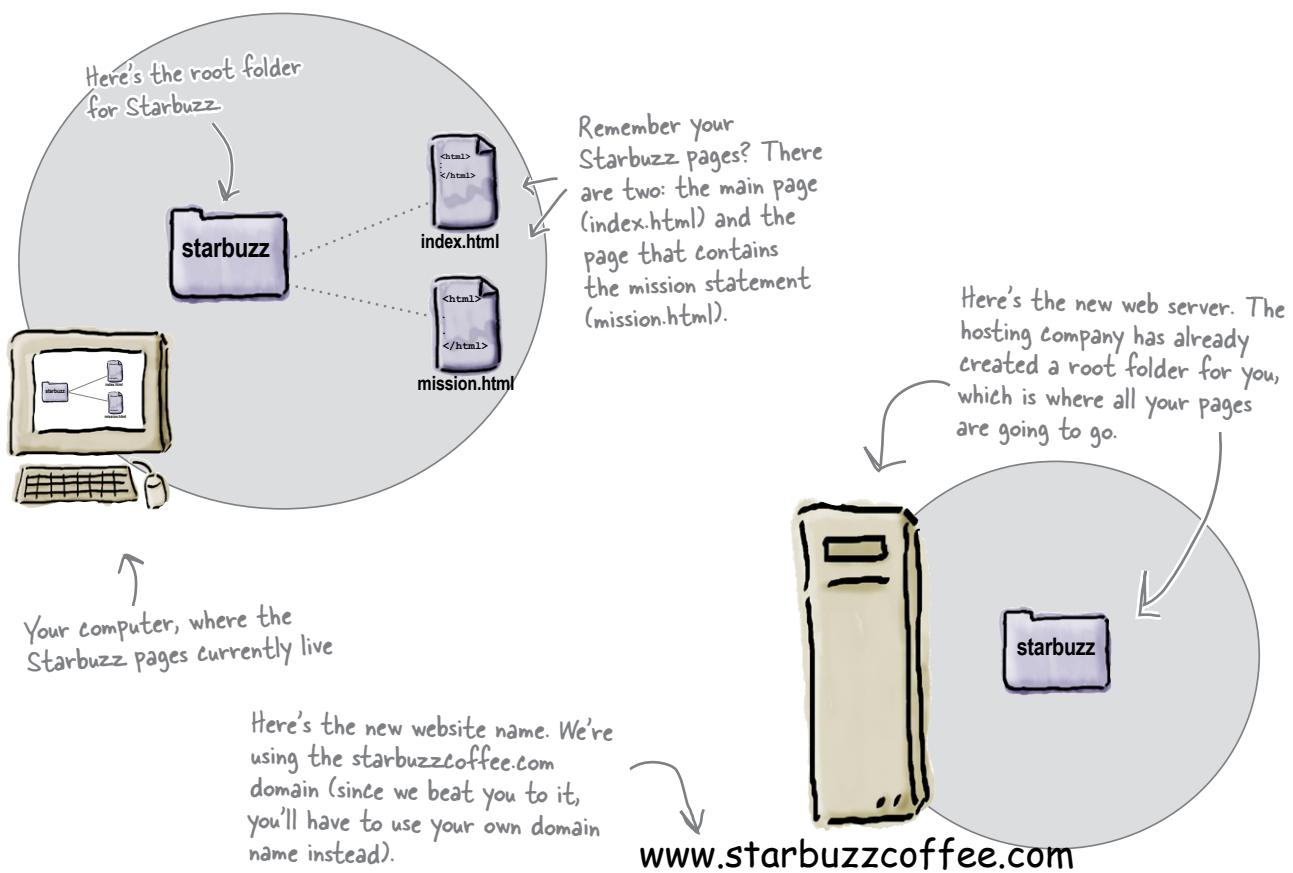
My Domain Name:



Moving in

Congratulations! You've got your hosting company lined up, you've found a domain name, and you've got a server all ready for your web pages. (Even if you don't, keep following along because this is important stuff.)

Now what? Well, it's time to move in, of course. So, take that For Sale sign down and gather up all those files; we're going to get them moved to the new server. Like any move, the goal is to get things moved from, say, the kitchen of your old place to the kitchen of your new place. On the Web, we're just worried about getting things from your own root folder to the root folder on the web server. Let's get back to Starbuzz and step through how we do this. Here's what things look like now:



there are no
Dumb Questions

Q: Wait a sec, what's the “root folder” again?

A: Up until now, the root folder has just been the top-level folder for your pages. On the web server, the root folder becomes even more important because anything inside the root folder is going to be accessible on the Web.

Q: My hosting company seems to have called my root folder “`mydomain_com`”. Is that a problem?

A: Not at all. Hosting companies call root folders lots of different things. The important thing is that you know where your root folder is located on the server, and that you can copy your files to it (we'll get to that in a sec).

Q: So let me make sure I understand. We've been putting all our pages for the site in one folder, which we call the root folder. Now we're going to copy all that over to the server's root folder?

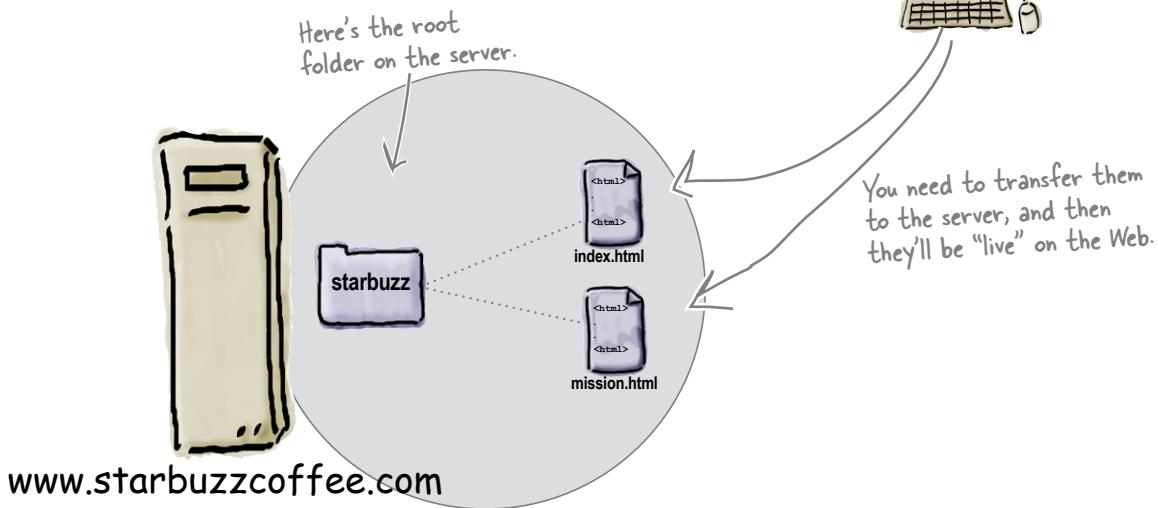
A: Exactly. You're going to take all the pages on your own computer, and put them all inside your site's root folder on the hosting company server.

Q: What about subfolders, like the “images” folder? Do I copy those too?

A: Yes, you're basically going to replicate all the pages, files, and folders in your own root folder onto the server. So if you've got an “images” folder on your computer, you'll have one on the server too.

Getting your files to the root folder

You're now one step away from getting Starbuzz Coffee on the Web: you've identified the root folder on your hosting company's server and all you need to do is copy your pages over to that folder. But how do you transfer files to a web server? There are a variety of ways, but most hosting companies support a method of file transfer called FTP, which stands for File Transfer Protocol. You'll find a number of applications out there that will allow you to transfer your files via FTP; we'll take a look at how that works on the next page.



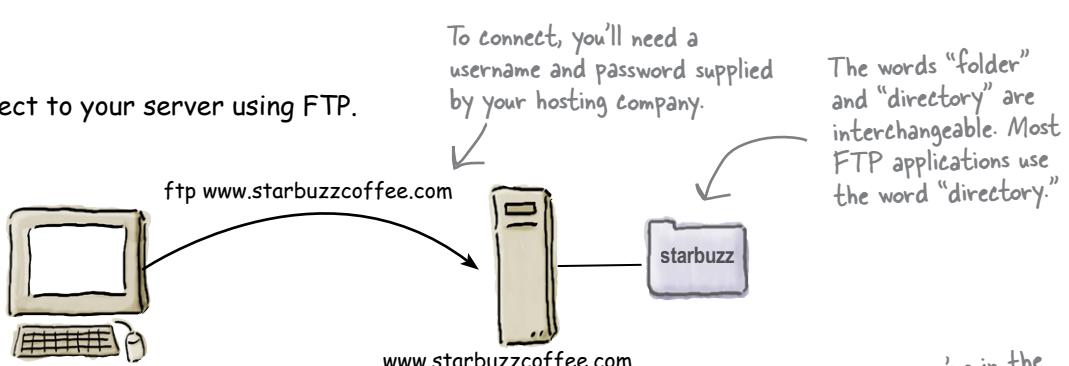


As much FTP as you can possibly fit in two pages

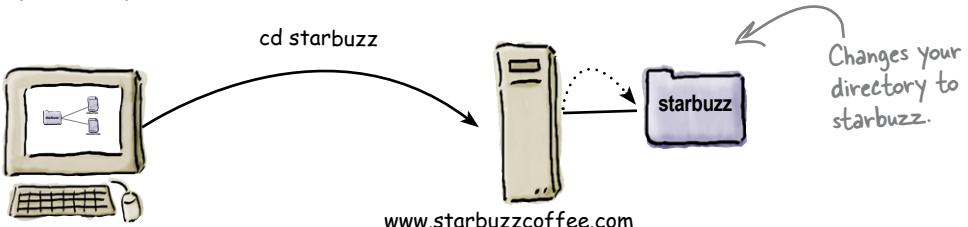
Seriously, this really is an HTML and CSS book, but we didn't want to leave you up a creek without a paddle. So, here's a very quick guide to using FTP to get your files on the Web. Keep in mind your hosting company might have a few suggestions for the best way to transfer your files to their servers (and since you are paying them, get their help). After the next few pages, we're off our detour and back to HTML and CSS until we reach the end of the book (we promise).

We'll assume you've found an FTP application. Some are command-line driven, some have complete graphical interfaces, and some are even built into applications like Dreamweaver and Expression Web. They all use the same commands, but with some applications you type them in yourself, while in others you use a graphical interface. Here's how FTP works from 10,000 feet:

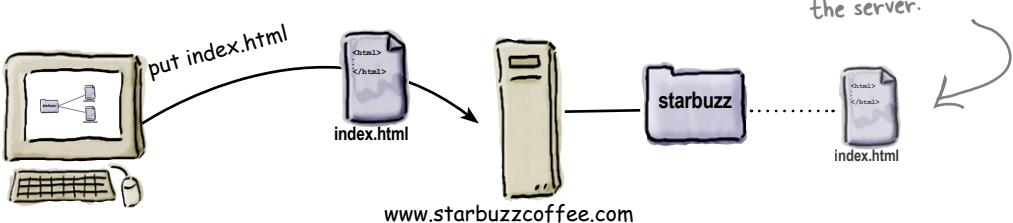
- ① First, connect to your server using FTP.



- ② Use the "cd" command to change your current directory to the directory where you want to transfer files.

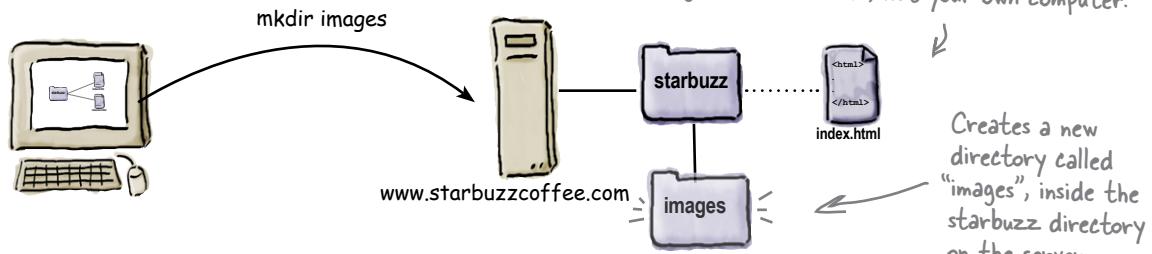


- ③ Transfer your files to the server using the "put" command.



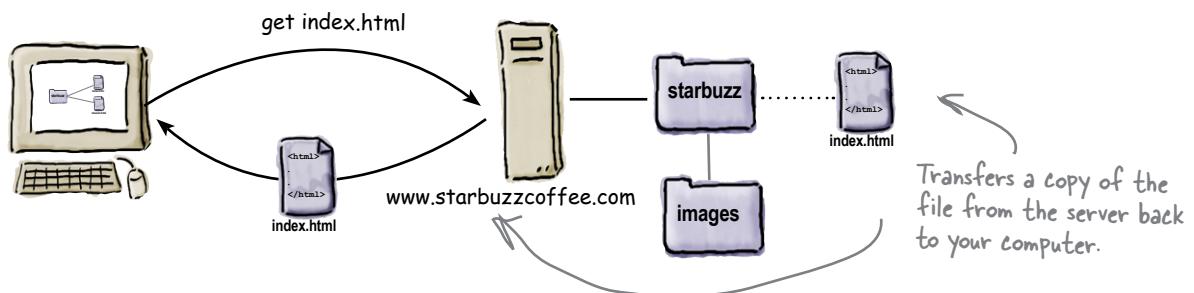


- ④ You can also make a new directory on the server with the "mkdir" command.



This is just like making a new folder, only you're doing it on the server, not your own computer.

- ⑤ You can retrieve files too, with the "get" command.



Let's put all that together. Here's an example of FTP being used from a command-line application:

```

File Edit Window Help Jam
%ftp www.starbuzzcoffee.com
Connected to www.starbuzzcoffee.com
Name: headfirst
Password: *****
230 User headfirst logged in.
ftp> dir
drwx----- 4096 Sep  5 15:07 starbuzz
ftp> cd starbuzz
CWD command successful
ftp> put index.html
Transfer complete.
ftp> dir
-rw----- 1022 Sep  5 15:07 index.html
ftp> mkdir images
Directory successfully created
ftp> cd images
CWD command successful
ftp> bye
    
```

Annotations for the terminal window:

- "Connect and log in." points to the connection line.
- "Get a directory of what is there." points to the 'dir' command output.
- "One directory called starbuzz." points to the 'starbuzz' directory entry.
- "Change to the starbuzz directory." points to the 'cd starbuzz' command.
- "Transfer index.html there." points to the 'put index.html' command.
- "Look at the directory; there's index.html." points to the second 'dir' command output.
- "Make a directory for images, and then quit using the bye command." points to the final 'bye' command.

Most FTP applications come with much friendlier graphical interfaces, so feel free to skip right over this if you're using one of those.

FTP commands

Whether you're typing in FTP commands on the command line, or using an FTP application with a graphical interface, the commands or operations you can perform are pretty much the same.

- *dir*: get a listing of the current directory.
- *cd*: change to another directory. ".." means up one directory here, too.
- *pwd*: display the current directory you're in.
- *put <filename>*: transfers the specified filename to the server.
- *get <filename>*: retrieves the specified filename from the server, back to your computer.

there are no Dumb Questions

Q: My hosting company told me to use SFTP, not FTP. What's the difference?

A: SFTP, or Secure File Transfer Protocol, is a more secure version of FTP, but works mostly the same way. Just make sure your FTP application supports SFTP before you make a purchase.

Q: So do I edit my files on my computer and then transfer them each time I want to update my site?

A: Yes, for small sites, that is normally the way you do things. Use your computer to test your changes and make sure things are working the way you want before transferring your files to the server. For larger websites, organizations often create a test site and a live site so that they can preview changes on the test site before they are moved to the live site.

If you're using a tool like Dreamweaver or Coda, these tools will allow you to test your changes on your own computer, and then when you save your files, they are automatically transferred to the website.

Q: Can I edit my files directly on the web server?

A: That usually isn't a good idea because your visitors will see all your changes and errors before you have time to preview and fix them.

That said, some hosting companies will allow you to log into the server and make changes on the server. To do that, you usually need to know your way around a DOS or Linux command prompt, depending on what kind of operating system your server is running.



Popular FTP applications

Here are a few of the most popular FTP applications for Mac and Windows:

For Mac OS X:

- Fetch (<http://fetchsoftworks.com/>) is one of the most popular FTP applications for Mac. \$
- Transmit (<http://www.panic.com/transmit/>). \$. \$
- Cyberduck (<http://cyberduck.ch/>). FREE

For Windows:

- Smart FTP (<http://www.smartftp.com/download/>). \$
- WS_FTP (<http://www.wsftple.com/>). FREE for the basic version, \$ for the Pro version
- Cyberduck (<http://cyberduck.ch/>). FREE

Most FTP applications have a trial version you can download to try before you buy.



DO try this at home

It's another homework assignment for you (check each item as you do it):

- Make sure you know where your root folder is on the server at your hosting company.
- Figure out the best way (and the best tool to use) to transfer files from your computer to the server.
- For now, go ahead and transfer the Starbuzz "index.html" and "mission.html" files to the root folder of the server.

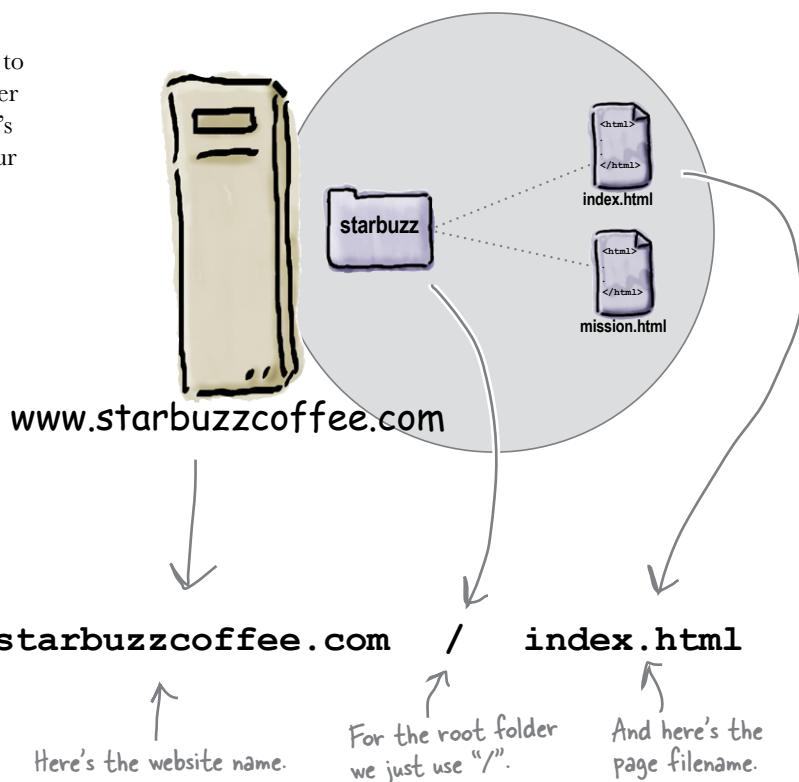


End of Web Detour

Back to business...

That's the end of the detour, and we're back on the web superhighway. At this point, you should have the two Starbuzz pages, "index.html" and "mission.html", sitting under your root folder on a server (or if not, you're at least following along).

After all this work, wouldn't it be satisfying to make your browser retrieve those pages over the Internet and display them for you? Let's figure out the right address to type into your browser...



URL Mainstreet, USA

You've probably heard the familiar "h" "t" "t" "p" "colon" "slash" "slash" a zillion times, but what does it mean? First of all, the web addresses you type into the browser are called *URLs* or Uniform Resource Locators.

If it were up to us, we would have called them "web addresses," but no one asked, so we're stuck with Uniform Resource Locators. Here's how to decipher a URL:

`http://www.starbuzzcoffee.com/index.html`

The first part of the URL tells you the protocol that needs to be used to retrieve the resource.

The second part is the website name. At this point, you know all about that.

And the third part is the absolute path to the resource from the root folder.

To locate anything on the Web, as long as you know the server that hosts it, and an *absolute path* to the resource, you can create a URL and most likely get a web browser to retrieve it for you using some *protocol*—usually HTTP.

A Uniform Resource Locator (URL) is a global address that can be used to locate anything on the Web, including HTML pages, audio, video, and many other forms of web content.

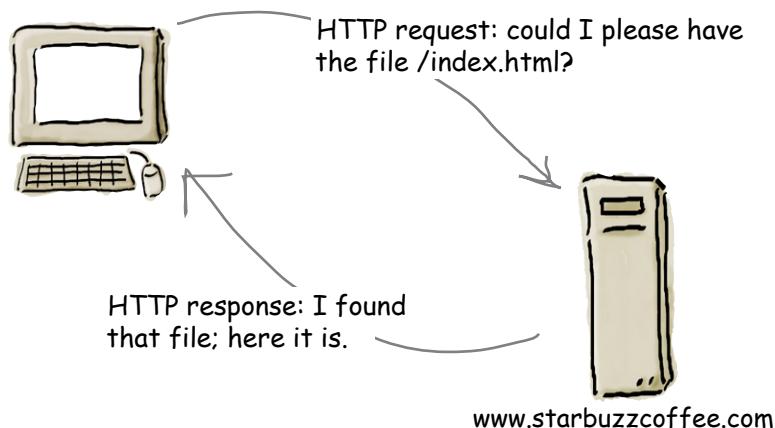
In addition to specifying the location of the resource, a URL also names the protocol that you can use to retrieve that resource.



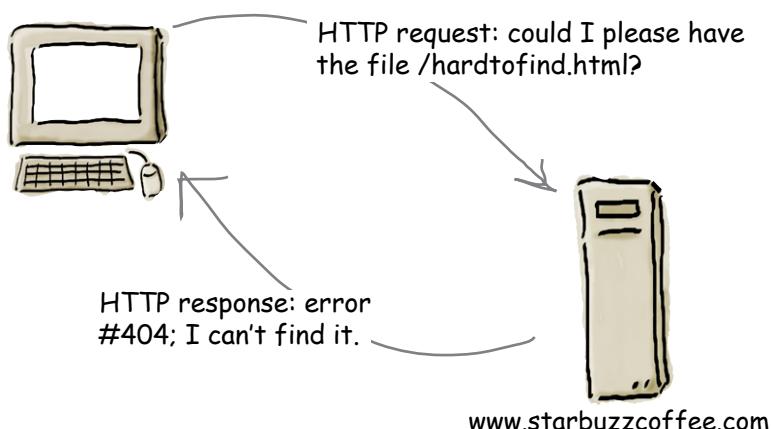
What is HTTP?

HTTP is also known as the *HyperText Transfer Protocol*. In other words, it's an agreed-upon method (a protocol) for transferring hypertext documents around the Web. While "hypertext documents" are usually just HTML pages, the protocol can also be used to transfer images, or any other file that a web page might need.

HTTP is a simple request and response protocol. Here's how it works:



So each time you type a URL into your browser's address bar, the browser asks the server for the corresponding resource using the HTTP protocol. If the server finds the resource, it returns it to the browser and the browser displays it. What happens if the server doesn't find it?



If the resource can't be found, you'll get the familiar "404 Error," which the server reports back to your browser.

Whatever you do,
don't pronounce URL as
"Earl," because that's my
name. It's pronounced
U-R-L.

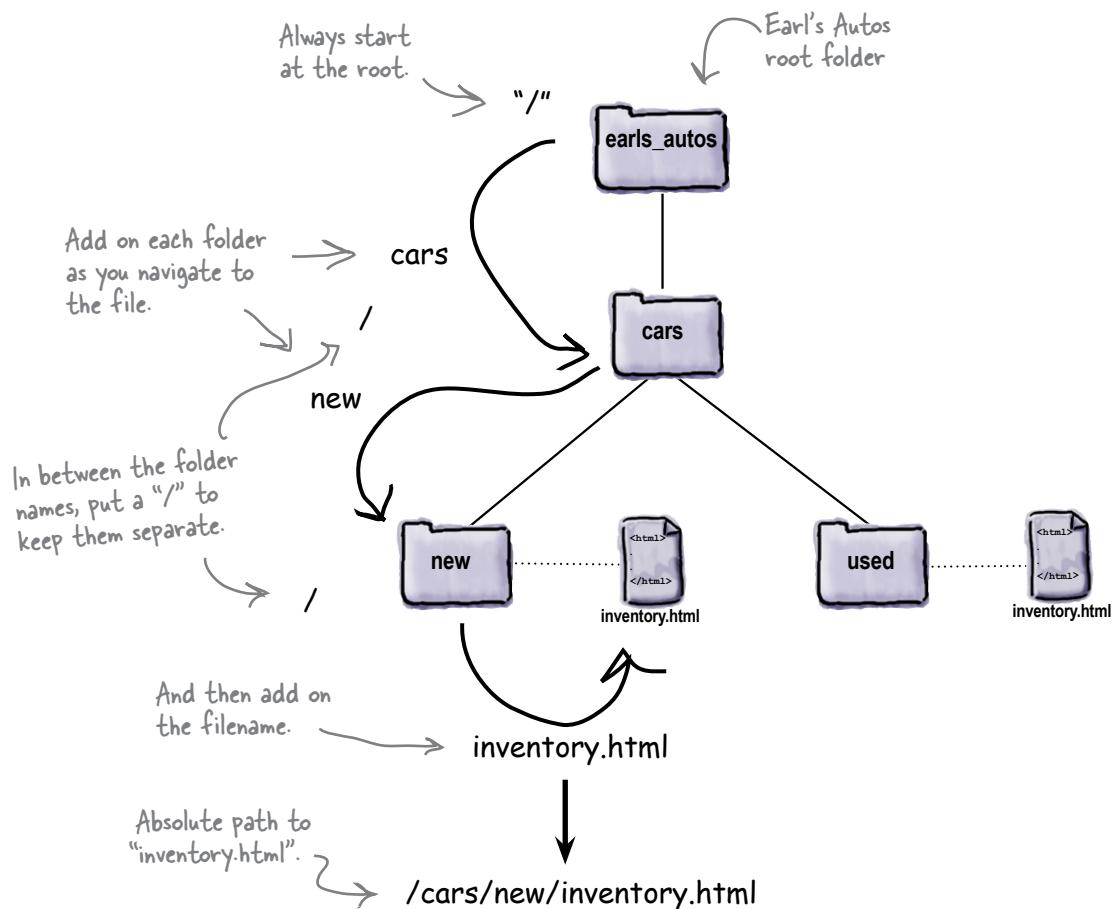


What's an absolute path?

The last time we talked about paths, we were writing HTML to make links with the `<a>` element. The path we're going to look at now is the absolute path part of a URL, the last part that comes after the protocol (http) and the website name (www.starbuzzcoffee.com).

An absolute path tells the server how to get from your root folder to a particular page or file. Take Earl's Autos site, for example. Say you want to look in Earl's inventory to see if your new Mini Cooper has come in. To do that, you'll need to figure out the absolute path to the file "inventory.html" that is in the "new" folder. All you have to do is trace through the folders, starting at the root, to get to the "new" folder where his "inventory.html" file is located. The path is made up of all the folders you go through to get there.

So, that looks like root (we represent root with a "/"), "cars", "new", and finally, the file itself, "inventory.html". Here's how you put that all together:



there are no
Dumb Questions

Q: What is important about the absolute path?

A: The absolute path is what a server needs to locate the file you are requesting. If the server didn't have an absolute path, it wouldn't know where to look.

Q: I feel like I understand the pieces (protocols, servers, websites, and absolute paths), but I'm having trouble connecting them.

A: If you add all those things together, you have a URL, and with a URL you can ask a browser to retrieve a page (or other kinds of resources) from the Web. How? The protocol part tells the browser the method it should use to retrieve the resource (in most cases, this is HTTP). The website part (which consists of the server name and the domain name) tells the browser which computer on the Internet to get the resource from. And the absolute path tells the server what page you're after.

Q: We learned to put relative paths in the href attribute of our <a> elements. How can the server find those links if they aren't absolute?

A: Wow, great question. When you click on a link that is relative, behind the scenes the browser creates an absolute path out of that relative path and the path of the page that you click on. So, all the web server ever sees are absolute paths, thanks to your browser.

Q: Would it help the browser if I put absolute paths in my HTML?

A: Ah, another good question, but hold that thought—we'll get back to that in a sec.



Sharpen your pencil

You've waited long enough. It's time to give your new URL a spin. Before you do, fill in the blanks below and then type in the URL (like you haven't already). If you're having any problems, this is the time to work with your hosting company to get things sorted out. If you haven't set up an hosting company, fill in the blanks for www.starbuzzcoffee.com, and type the URL into your browser anyway.

::/

protocol

website name

absolute path



I'd like my visitors to be able to type "http://www.starbuzzcoffee.com" and not have to type the "index.html". Is there a way to do that?

Yes, there is. One thing we haven't talked about is what happens if a browser asks for a directory rather than a file from a web server. For instance, a browser might ask for:

`http://www.starbuzzcoffee.com/images/`

or

`http://www.starbuzzcoffee.com/`

Remember, when we're talking about web servers or FTP, we usually use the term "directory" instead of "folder." But they're really the same thing.

The images directory in the root directory

The root directory itself

When a web server receives a request like this, it tries to locate a *default* file in that directory. Typically a default file is called "index.html" or "default.htm" and if the server finds one of these files, it returns the file to the browser to display.

So, to return a file by default from your root directory (or any other directory), just name the file "index.html" or "default.htm".

But you need to find out what your hosting company wants you to name your default file, because it depends on the type of server they use.

But I asked about "http://www.starbuzzcoffee.com", which looks a little different. It doesn't have the ending "/".

Oops, you sure did. When a server receives a request like yours without the trailing "/" and there is a directory with that name, then the server will add a trailing slash for you. So if the server gets a request for:

`http://www.starbuzzcoffee.com`

it will change it to:

`http://www.starbuzzcoffee.com/`

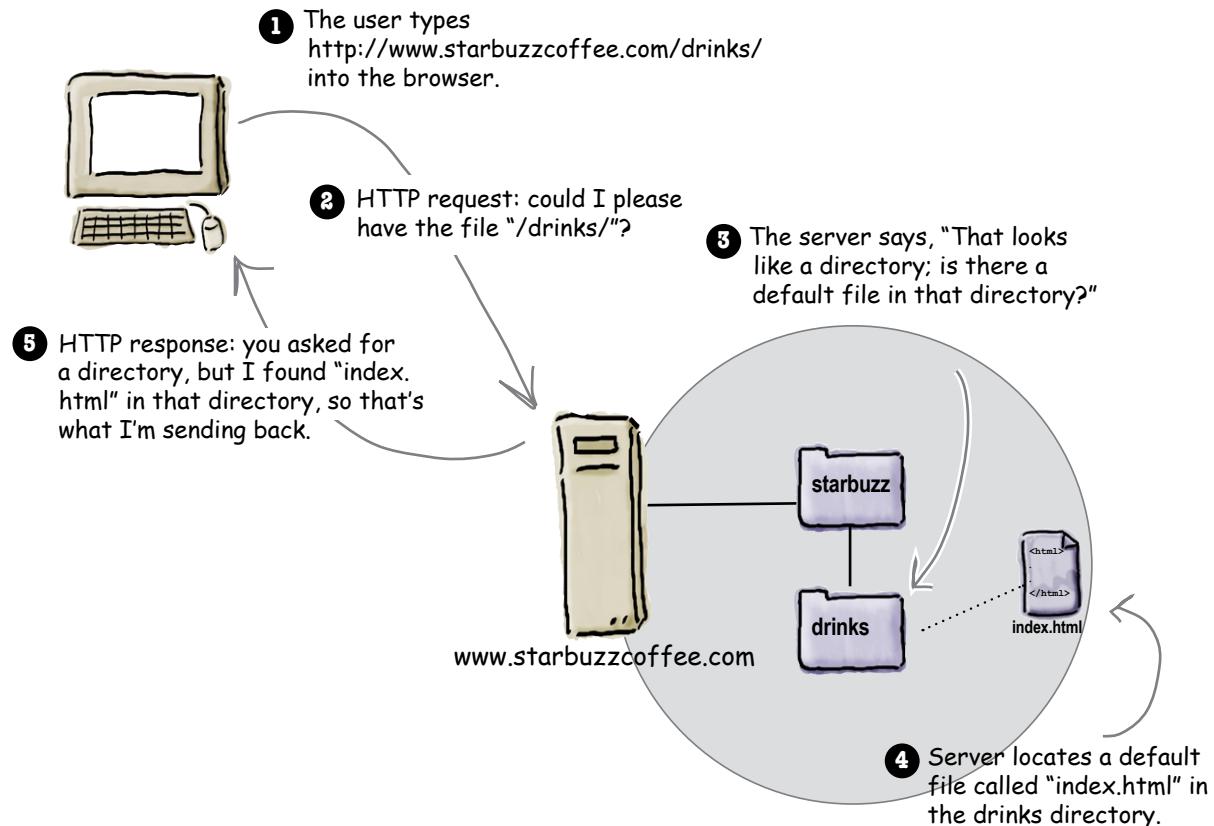
which will cause the server to look for a default file, and in the end it will return the file as if you'd originally typed:

`http://www.starbuzzcoffee.com/index.html`



How default pages work

Behind
the Scenes



there are no
Dumb Questions

Q: So anyone who comes to my site with the URL `http://www.mysite.com` is going to see my "index.html" page?

A: Right. Or, possibly "default.htm" depending on which kind of web server your hosting company is using. (Note that "default.htm" usually has no "l" on the end. This is a Microsoft web server oddity.)

There are other possible default filenames, like "index.php", that come into play if you start writing scripts to generate your pages. That's way beyond this book, but that doesn't mean you won't be doing it in the future.

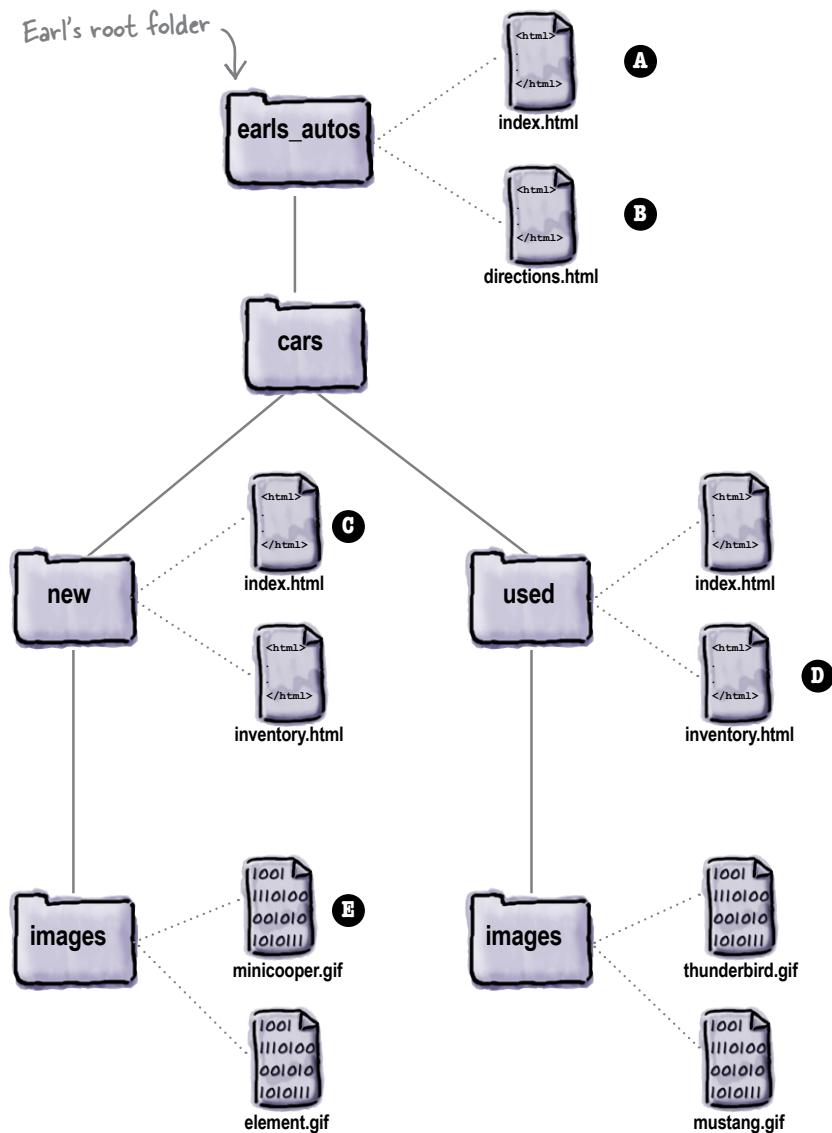
Q: So when I'm giving someone my URL, is it better to include the "index.html" part or not?

A: Not. It's always better to leave it off. What if, in the future, you change to another web server and it uses another default file name like "default.htm"? Or you start writing scripts and use the name "index.php"? Then the URL you originally gave out would no longer be valid.



Earl needs a little help with his URLs

Earl may know Earl, but he doesn't know U-R-L. He needs a little help figuring out the URL for each of the files below, labeled A, B, C, D, and E. On the right, write in the URL needed to retrieve each corresponding file from www.earlsautos.com.







How do we link to other websites?

URLs aren't just for typing into browsers; you can use them right in your HTML. And, of course, right on cue, the Starbuzz CEO has a new task for you: make a link from the main Starbuzz page over to the caffeine information at `http://wickedlysmart.com/buzz`. As you can probably guess, we're going to throw that URL right into an `<a>` element. Here's how:

```
<a href="http://wickedlysmart.com/buzz">Caffeine Buzz</a>
```

An everyday, normal, garden-variety `<a>` element.

We've put a URL in the `href`. Clicking on the label "Caffeine Buzz" will retrieve a page from `wickedlysmart.com/buzz`.

That's all there is to it. To link to any resource on the Web, all you need is its Uniform Resource Locator, which goes in the `<a>` element as the value of the `href` attribute. Let's go ahead and add this in the Starbuzz "index.html" page.

Linking to Caffeine Buzz

Open your Starbuzz “index.html” file in the “chapter4/starbuzz” folder, and scan down to the bottom. Let’s add two new links: a relative link to the mission statement in “mission.html”, and a link to Caffeine Buzz. Make the changes below, then save and load your “index.html” file in your browser. Click on the link and enjoy the Caffeine Buzz.

```

<html>
  <head>
    <title>Starbuzz Coffee</title>
    <style type="text/css">
      body {
        background-color: #d2b48c;
        margin-left: 20%;
        margin-right: 20%;
        border: 2px dotted black;
        padding: 10px 10px 10px 10px;
        font-family: sans-serif;
      }
    </style>
  </head>

  <body>
    <h1>Starbuzz Coffee Beverages</h1>
    <h2>House Blend, $1.49</h2>
    <p>A smooth, mild blend of coffees from Mexico,  
Bolivia and Guatemala.</p>

    <h2>Mocha Cafe Latte, $2.35</h2>
    <p>Espresso, steamed milk and chocolate syrup.</p>

    <h2>Cappuccino, $1.89</h2>
    <p>A mixture of espresso, steamed milk and foam.</p>

    <h2>Chai Tea, $1.85</h2>
    <p>A spicy drink made with black tea, spices,  
milk and honey.
    </p>
    <p>
      <a href="mission.html">Read about our Mission</a>.
      <br>
      Read the <a href="http://wickedlysmart.com/buzz">Caffeine Buzz</a>.
    </p>
  </body>
</html>

```

And we've added some structure here by grouping the links and text into a paragraph.

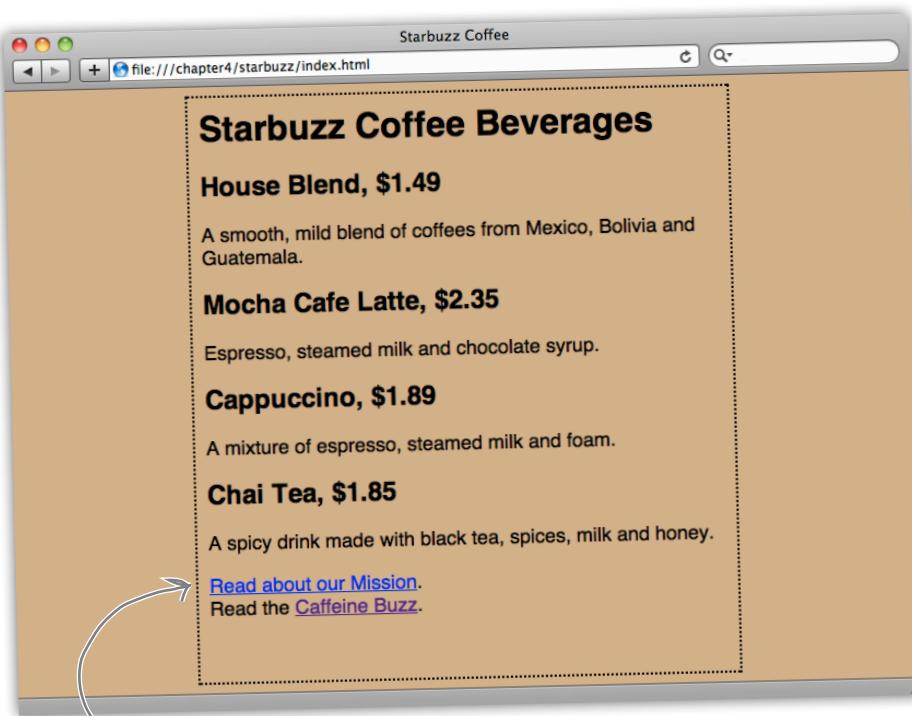
Here's the link to the "mission.html" file. This uses a relative path to link to "mission.html".

We added a `
` to put the links on two different lines.

Here's where we've added the link to the wickedlysmart.com/buzz page.

And now for the test drive...

Here's the page with the new link, just as we planned.

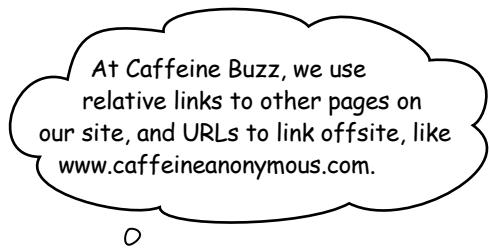


Here's the new link. Notice, we only linked the words "Caffeine Buzz," so it looks a little different from the other link.

And when you click on the link, your browser will make an HTTP request to wickedlysmart.com/buzz and then display the result.



there are no Dumb Questions



Q: It seems like there are two ways to link to pages now: relative paths and URLs.

A: Relative paths can only be used to link to pages within the same website, while URLs are typically used to link to other websites.

Q: Wouldn't it be easier if I just stuck with URLs for links to my own pages and outside pages? That would work, wouldn't it?

A: Sure, it would work, but there's a couple of reasons you don't want to go there. One problem is that URLs are hard to manage when you have a lot of them in a web page: they're long, difficult to edit, and they make HTML more difficult to read (for you, the page author).

Also, if you have a site with nothing but URLs that link to local pages and you move the site or change its name, you have to go change all those URLs to reflect the new location. If you use relative paths, as long as your pages stay in the same set of folders—because the links are all relative—you don't have to make any changes to your `<a>` element `href` attributes.

So, use relative links to link to your own pages in the same site, and URLs to link to pages at other sites.

Q: Haven't we seen one other protocol? I kept seeing "file:/// before we started using a web server.

A: Yes; good catch. The file protocol is used when the browser is reading files right off your computer. For example, the file URL, "file:///chapter4/starbuzz/index.html", tells the browser that the file "index.html" is located at the path "/chapter4/starbuzz/". This path may look different depending on your operating system and browser.

One important thing to notice in case you try to type in a file URL is that the file URL has three slashes, not two, like HTTP. Remember it this way: if you take an HTTP URL and delete the website name you'll have three slashes, too.

Q: Are there other protocols?

A: Yes, many browsers can support retrieval of pages with the FTP protocol, and there is a mail protocol that can send data via email. HTTP is the protocol you'll be using most of the time.

Q: I've seen URLs that look like this: <http://www.mydomain.com:8000/index.html>. Why is there a ":8000" in there?

A: The ":8000" is an optional "port" that you can put in an HTTP URL. Think of a port like this: the website name is like an address, and the port is like a mailbox number at an address (say, in an apartment complex). Normally everything on the Web is delivered to a default port (which is 80), but sometimes web servers are configured to receive requests at a different port (like 8000). You'll most likely see this on test servers. Regular web servers almost always accept requests on port 80. If you don't specify a port, it defaults to 80.

Five-Minute Mystery



The Case of Relatives and Absolutes

PlanetRobots, Inc., faced with the task of developing a website for each of its two company divisions—PlanetRobot Home and PlanetRobot Garden—decided to contract with two firms to get the work done. RadWebDesign, a seemingly experienced firm, took on the Home division's website and proceeded to write the site's internal links using only URLs (after all, they're more complicated, so they must be better). A less experienced, but well-schooled firm, CorrectWebDesign, was tasked with PlanetRobot's Garden site, and used relative paths for links between all the pages within the site.

Just as both projects neared completion, PlanetRobots called with an urgent message: "We've been sued for trademark infringement, so we're changing our domain name to RobotsRUs. Our new web server is going to be www.robotsrus.com." CorrectWebDesign made a couple of small changes that took all of five minutes and was ready for the site's unveiling at the RobotsRUs corporate headquarters. RadWebDesign, on the other hand, worked until 4 a.m. to fix their pages but luckily completed the work just in time for the unveiling. However, during a demo at the unveiling, the horror of horrors occurred: as the team leader for RadWebDesign demonstrated the site, he clicked on a link that resulted in a "404—Page Not Found" error. Displeased, the CEO of RobotsRUs suggested that RadWebDesign might want to consider changing their name to BadWebDesign and asked CorrectWebDesign if they were available to consult on fixing the Home site.

What happened? How did RadWebDesign flub things up so badly when all that changed was the name of the web server?

Web page fit and finish

Can you say “web career”? You’ve certainly delivered everything the Starbuzz CEO has asked for, and you’ve now got a high-profile website under your belt (and in your portfolio).

But you’re not going to stop there. You want your websites to have that professional “fit and finish” that makes good sites into great ones. You’re going to see lots of ways to give your sites that extra “polish” in the rest of this book, but let’s start here with a way to improve your links.

Improving accessibility by adding a title to your links

Wouldn’t it be nice if there were a way to get more information about the link you’re about to click on? This is especially important for the visually impaired using screen readers because they often don’t want the entire URL spoken to them: (“h” “t” “t” p” “:” “slash” “slash” “w” “w” “w” “dot”), and yet the link’s label usually only gives a limited description, like “Caffeine Buzz.”

The `<a>` element has an attribute called `title` just for this purpose. Some people are confused by this attribute name because there’s an *element* named `<title>` that goes in the `<head>`. They have the same name because they are related—it is often suggested that the value of the `title` attribute be the same as value of the `<title>` element of the web page you are linking to. But that isn’t a requirement and often it makes more sense to provide your own, more relevant description in the `title` attribute.

Here’s how you add a `title` attribute to the `<a>` element:

```
Read the <a href="http://wickedlysmart.com/buzz"
          title="Read all about caffeine on the Buzz">Caffeine Buzz</a>
```



The `title` element has a value that is a textual description of the page you are linking to.

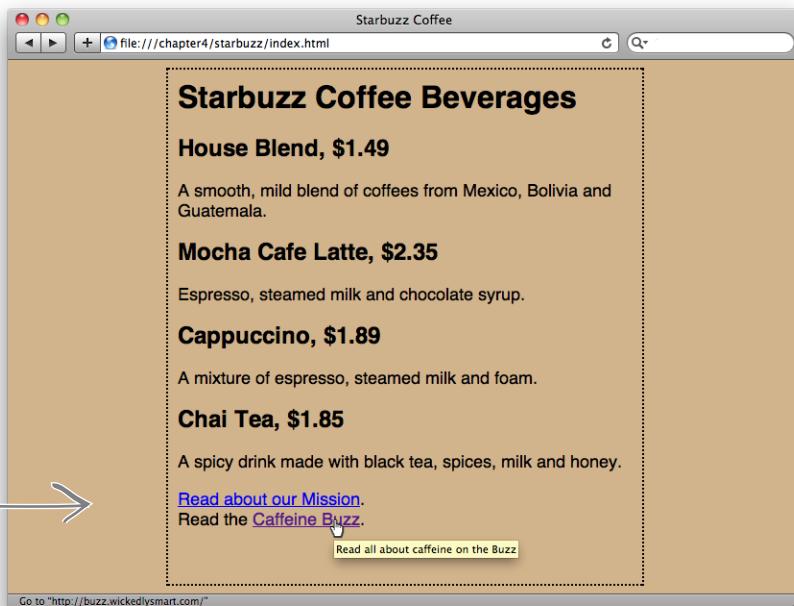


Now that we’ve got a `title` attribute, let’s see how your visitors would make use of it. Different browsers make different use of the title, but many display a tool tip. Add the changes above to your “index.html” file and reload the page to see how it works in your browser.

The title test drive...

For most browsers, the title is displayed as a tool tip when you pass the mouse over a link. Remember that browsers for the visually impaired may read the link title aloud to a visitor.

The title is displayed as a tool tip in most browsers. Just pass your mouse over the link and hold it there a second to see the tool tip.



The Head First Guide to Better Links

Here are a few tips to keep in mind to further improve the fit and finish of your links:



- ♦ *Keep your link labels concise.* Don't make entire sentences or large pieces of text into links. In general, keep them to a few words. Provide additional information in the title attribute.
- ♦ *Keep your link labels meaningful.* Never use link labels like "click here" or "this page." Users tend to scan pages for links first, and then read pages second. So, providing meaningful links improves the usability of your page. Test your page by reading just the links on it; do they make sense? Or do you need to read the text around them?
- ♦ *Avoid placing links right next to each other;* users have trouble distinguishing between links that are placed closely together.



Open your Starbuzz “index.html” file and add a title to the link to “mission.html” with the text “Read more about Starbuzz Coffee’s important mission.” Notice that we didn’t make the mission link’s label as concise as it should be. Shorten the link label to “our Mission.” Check the back of the chapter for the answer, and test your changes.



Linking into a page

So far, whenever you’ve linked to another page, the page loads and your browser displays it from the top.

But the CEO’s asking you to *link into* a particular spot in the page: the Coffee section.

Sound impossible? Come on, this is Head First—we’ve got the technology. How? Well, we haven’t told you everything about the `<a>` element yet. Turns out the `<a>` element can team up with the `id` attribute to take you straight to a specific point in a page.

Sources

One common source of caffeine is the coffee plant, the beans from which are used to produce coffee. Caffeine content varies substantially between Arabica and Robusta species and to a lesser degree between varieties of each species.

One dose of caffeine is generally considered to be 100 mg. In theory, a single serving (6 fl oz / 150 ml) of drip coffee or one-half caffeine tablet would deliver this dose. In the real world, coffee varies considerably in caffeine content per serving, ranging from about 75 mg to 250 mg. Generally, dark roast coffee has less caffeine than lighter roasts since the roasting process reduces caffeine content of the bean.

Tea is another common source of caffeine in many cultures. Tea contains somewhat less caffeine per serving than coffee, (usually about half as much, depending on the strength of the brew), though certain types of tea, such as Lapsang sou chong smoked teas, and oolong contain more caffeine.

Caffeine is also common in soft drinks such as cola. Such drinks typically contain about 25 mg to 50 mg of caffeine per serving. Some “energy drinks” such as Red Bull contain 80 mg, while others offer considerably more caffeine per serving, from 100 mg to 400 mg.

Mateine and guaranine are other names for **caffeine**. The names come from yerba mate and guarana respectively, caffeine-containing plants used for tea and other things. Many yerba mate enthusiasts insist that mateine is a stereoisomer of caffeine and thus a different substance altogether. However, this is impossible: caffeine is an achiral molecule with no chiral centers, and therefore has no stereoisomers. Similar claims are sometimes made of guaranine.

Caffeine is sometimes called **theine** when it is found in tea, as the caffeine in tea was once thought to be a separate compound to the caffeine found in coffee. But tea does contain another xanthine, theophylline whose chemical structure is C7HgN4O2 compared to caffeine's C8H10N4O2. This is similar to the naming problem with **mateine** and **guaranine**.

Coffee

All fluid ounces are U.S. fluid ounces.

- Coffee, brewed (drip) - 4 to 20 mg/fl oz (130 to 680 mg/litre) (40 to 170 mg/5 fl oz)
- Coffee, decaffeinated - 0.4 to 0.6 mg/fl oz (13 to 20 mg/litre)
- Coffee, instant - 4 to 12 mg/fl oz (130 to 400 mg/litre)
- Espresso Arabica - ~40 mg/fl oz (1360 mg/litre)
- Espresso Robusta - ~100 mg/fl oz (3400 mg/litre)

Teas and other infusions

Using the id attribute to create a destination for <a>

We haven't talked about the `id` attribute yet; it's an important attribute with special properties, and we'll get into more detail about other special properties of `id` later in the book. For now, think of it as a way of uniquely identifying an element. One special property that elements with `ids` get is that you can link to them. Let's see how to use the `id` attribute to create a destination in a page for `<a>`.

- ➊ Find the location in the page where you'd like to create a landing spot. This can be any text on the page, but often is just a heading.
- ➋ Choose an identifier name for the destination, like "coffee" or "summary" or "bio," and insert an `id` attribute into the opening tag of the element.

Let's give it a try. Say you want to provide a way to link to the Chai Tea item on the Starbuzz page. Here's what it looks like now:

```
<h2>Chai Tea, $1.85</h2>
<p>A spicy drink made with black tea, spices, milk
and honey.</p>
```

Here's the snippet from "index.html" with the Chai heading and description.

Following the two steps above, we get this:

Add the `id` to the opening tag of the heading.

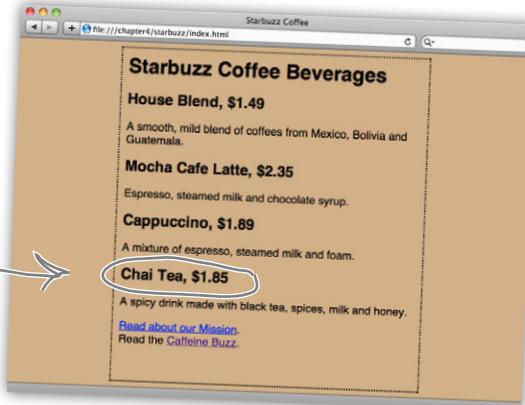
And we'll give this destination the identifier "chai".

It's important that your `id` be unique. That is, the "chai" `id` must be the only "chai" `id` in the page!

```
<h2 id="chai">Chai Tea, $1.85</h2>
```

```
<p>A spicy drink made with black tea, spices, milk and
honey.</p>
```

By giving it an `id`, you've made a destination out of the Chai Tea heading in the "index.html" page.



How to link to elements with ids

You already know how to link to pages using either relative links or URLs. In either case, to link to a specific destination in a page, just add a # on the end of your link, followed by the destination identifier. So if you wanted to link from any Starbuzz Coffee web page to the “chai” destination heading, you’d write your <a> element link this:

```
<a href="index.html#chai">See Chai Tea</a>
```

Unfortunately, linking to Chai Tea isn’t very impressive because the whole page is small enough that it easily fits in the browser. Let’s link to the Coffee section of <http://wickedlysmart.com/buzz> instead. Here’s what you’re going to do:

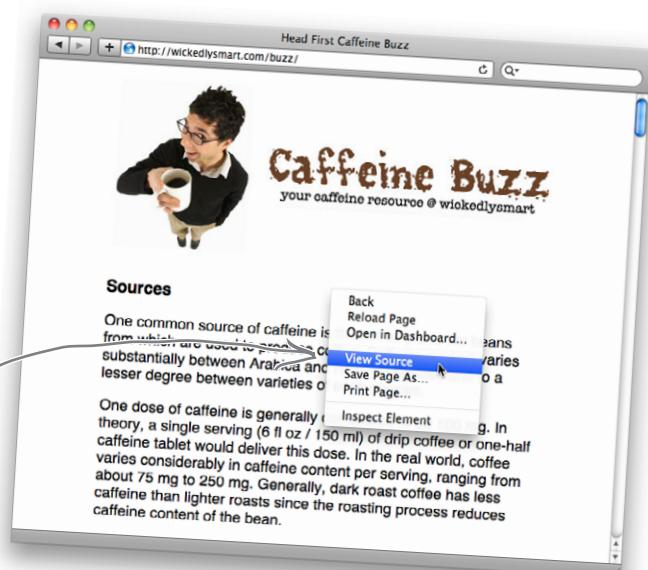
- ➊ Figure out the id of the Coffee heading.
- ➋ Alter the existing <a> element in the Starbuzz Coffee “index.html” file to point to the destination heading.
- ➌ Reload your “index.html” page and test out the link.

The main benefit of specific destinations is to link to locations in long files so your visitors don’t have to scroll through the file looking for the right section.

Finding the destination heading

To find the destination heading, you’re going to have to look at the wickedlysmart.com/buzz page and view their HTML. How? Almost all browsers have a “View Source” option. So, visit the page and when it is fully loaded, choose the “View Source” option, and you’ll see the markup for the page.

In most browsers, you can right-click to “View Source.”



Now that you've got your hands on their HTML...

Scroll down until you see the Coffee section; it looks like this:

```
...  
This is similar to the naming problem  
with <b>mateine</b> and <b>guaranine</b>.  
</p>  
  
<h3 id="Coffee">Coffee</h3> ←  
<p>  
<i>All fluid ounces are U.S. fluid ounces.</i>  
</p>
```

Just a small snippet from
the Caffeine Buzz page.

Here's the Coffee section. You can
see the heading for it along with
the start of the paragraph below.

Ahhh, and here is the
destination heading. It has
the name "Coffee".

Reworking the link in "index.html"

Now all you need to do is revisit the link to Caffeine Buzz
and add on the destination anchor name, like this:

This is a snippet from the
Starbuzz "index.html" file.

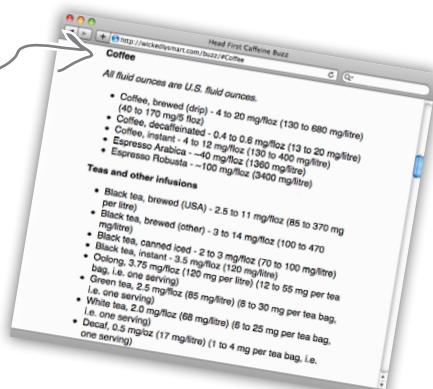
The default file at
wickedlysmart.com/buzz is
index.html. So, we'll add that
to the URL so we can use it
with the destination id.

Add # along with
the destination id
to your href.

```
Read the <a href="http://wickedlysmart.com/buzz/index.html#Coffee"  
title="Read all about caffeine on the Buzz">Caffeine Buzz</a>
```



Make this change to your Starbuzz "index.html" file.
Reload and click on the "Caffeine Buzz" link. You
should be taken directly to the Coffee section of
Caffeine Buzz's front page.



there are no Dumb Questions

Q: When I have two attributes in an element, is the order important? For example, should the title attribute always come after the href?

A: The order of attributes is not important in any element (if it were, we'd all have headaches 24/7). So, use any ordering you like.

Q: How would I create a tool tip for an element that's not an <a>?

A: You can add the title attribute to any element, so if you want a tool tip on, say, a heading, you can add a title attribute to your <h1> opening tag just like we did with <a>. There are a few elements that use the title attribute for more than just a tool tip, but the tool tip is its most common purpose.

Q: Can I add an id attribute to any element?

A: Yes, you can. You could link into the middle of a paragraph by adding an id to an element, for instance. It's unlikely that you'll often need to do that, but you can do it if you want.

Q: Could I link to a link by adding an id attribute to an <a> element in the destination?

A: Yes!

Q: I noticed in the id names, you used "chai" with all lowercase letters and Caffeine Buzz used "Coffee" with an uppercase "C". Does it matter?

A: You can use any combination of upper- and lowercase characters in your id attributes. Just make sure you are consistent and always use the same upper- and lowercase letters in your hrefs and destination id (which is why it is often easier to make these names entirely lowercase every time). If you aren't consistent, don't expect your links to work correctly on every browser. The most important thing about the id name you choose is that it must be unique in your page.

Q: Can I put a link to a destination from within the same document?

A: Sure. In fact, it is common to define a destination "top" at the top of a page (say, on the top heading of the page) and have a link at the bottom of the page reading "Back to top." It is also common in long documents to have a table of contents for the entire page. For instance, to link to the "top" destination heading in the same page, you would write Back to top.

Q: Why did we need to add the "/index.html" to the Buzz URL in order to create a link to the destination heading? Couldn't we have just written:
<http://wickedlysmart.com/buzz#Coffee>?

A: No, that won't always work because the browser adds that trailing slash on the end of the URL for you, which could end up replacing the id reference. You could, however, have written:
<http://wickedlysmart.com/buzz/#Coffee>, which will produce the same results as the link we created using "index.html". This will come in handy if you don't know if the default file is named "index.html".

Q: If a web page doesn't provide a destination and I still need to link to a specific part of the page, how can I?

A: You can't. If there is no destination (in other words, no element with an id), then you can't direct the browser to go to a specific location in a web page. You might try to contact the page author and ask them to add one (even better, tell them how!).

Q: Can I have a destination id like "Jedi Mindtrick" or does an id have to be only one word?

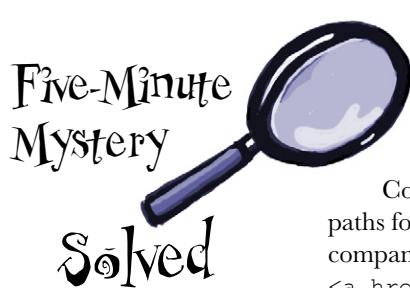
A: To work consistently with the most browsers, always start your id with a letter (A-Z or a-z) and follow it with any letter, digit, hyphen, underscore, colon, or period. So, while you can't use a space and have a name like "Jedi Mindtrick", that isn't much of a restriction because you can always have "Jedi-Mindtrick", "Jedi_Mindtrick", "JediMindtrick", and so on.

Q: How can I tell others what destinations they can link to?

A: There is no established way of doing this, and in fact, "View Source" remains the oldest and best technique for discovering the destinations you can link to.

Q: Do I always use just words as the content of an <a> element?

A: No. The <a> element has always been able to create links from words and images (inline content), and has recently been updated (in HTML5) so that you can create links from block elements, like <p> and <blockquote> too! So <a> can be used to create links from all kinds of things.



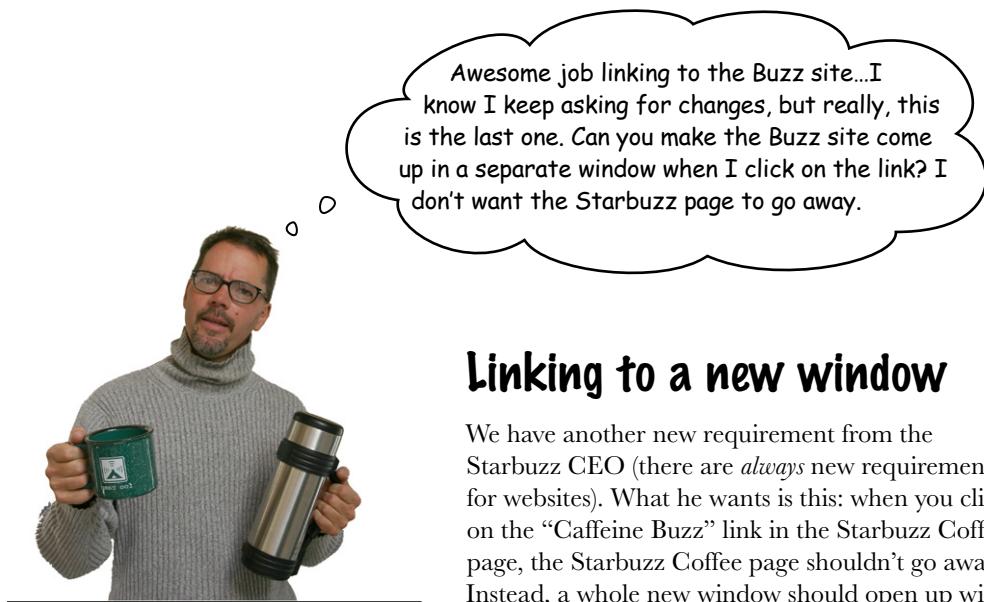
The Case of Relatives and Absolutes

So, how did RadWebDesign flub up the demo? Well, because they used URLs for their `hrefs` instead of relative links, they had to edit and change every single link from `http://www.planetrobots.com` to `http://www.robotsrus.com`. Can you say error-prone? At 3:00 a.m., someone yawned and accidentally typed `http://www.robotsru.com` (and as fate has it, that was the same link that the CEO clicked on at the demo).

Oops...
someone
forgot an "s"
on the end
of the name.

CorrectWebDesign, on the other hand, used relative paths for all internal links. For example, the link from the company's mission statement to the products page, ``, works whether the site is called PlanetRobots or RobotsRUs. So, all CorrectWebDesign had to do was update the company name on a few pages.

So RadWebDesign left the demo sleep-deprived and with a little egg on their face, while CorrectWebDesign left the meeting with even more business. But the story doesn't end there. It turns out that RadWebDesign dropped by a little coffeehouse/bookstore after the demo and, determined not to be outdone, picked up a certain book on HTML and CSS. What happened? Join us in a few chapters for "The Case of Brute Force Versus Style."



Linking to a new window

We have another new requirement from the Starbuzz CEO (there are *always* new requirements for websites). What he wants is this: when you click on the “Caffeine Buzz” link in the Starbuzz Coffee page, the Starbuzz Coffee page shouldn’t go away. Instead, a whole new window should open up with the Caffeine Buzz page in it, like this:

Here's the main Starbuzz Coffee page.

When the Caffeine Buzz window pops open, it will open over the top of the Starbuzz page, but the Starbuzz page will still be there.

What the CEO wants is a whole new window to open when you click on the Caffeine Buzz link.

Starbuzz Coffee Beverages

- House Blend, \$1.49**
A smooth, mild blend of coffee from Guatemala.
- Mocha Cafe Latte, \$2.29**
Espresso, steamed milk and chocolate sauce.
- Cappuccino, \$1.89**
A mixture of espresso, steamed milk and cinnamon.
- Chai Tea, \$1.85**
A spicy drink made with black tea and spices.

[Read about our Mission.](#) [Read the Caffeine Buzz.](#)

Head First Caffeine Buzz
your caffeine resource @ wickedlysmart

Sources

One common source of caffeine is the coffee plant, the beans from which are used to produce coffee. Caffeine content varies substantially between Arabica and Robusta species and to a lesser degree between varieties of each species.

One dose of caffeine is generally considered to be 100 mg. In theory, a single serving (6 fl oz / 150 ml) of drip coffee or one-half caffeine tablet would deliver this dose. In the real world, coffee varies considerably in caffeine content per serving, ranging from about 75 mg to 250 mg. Generally, dark roast coffee has less caffeine than lighter roasts since the roasting process reduces caffeine content of the bean.

Tea is another common source of caffeine in many cultures. Tea contains somewhat less caffeine per serving than coffee, (usually about half as much, depending on the strength of the brew), though certain types of

Opening a new window using target

To open a page in a new window, you need to tell the browser the name of the window in which to open it. If you don't tell the browser a specific window to use, the browser just opens the page in the *same* window. You can tell the browser to use a *different* window by adding a target attribute to the <a> element. The value of the target attribute tells the browser the "target window" for the page. If you use "_blank" for the target, the browser will *always* open a new window to display the page. Let's take a closer look:

```
<a target="_blank" href="http://wickedlysmart.com/buzz">  
    title="Read all about caffeine on the Buzz">Caffeine Buzz</a>
```

The target attribute tells the browser where to open the web page that is at the link in the href attribute. If there is no target, then the browser opens the link in the same window. If the target is "_blank", then the browser opens the link in a new window.

there are no
Dumb Questions



Exercise

Open your Starbuzz "index.html" file. Add the target attribute to the <a> tag that links to the Caffeine Buzz page. Now give it a try—did you get a new window?



Can you think of some advantages and some disadvantages to using the target attribute to open a page in a new window?

Q: I'm getting a new tab instead of a new window. Am I doing something wrong?

A: No, you're not. Most browsers now have a default setting to open new windows in a tab, rather than a whole new browser window, because that's what users seem to prefer. But a new tab and a new window are really the same thing; it's just that the tab shares the same window border as your original window. If you want to force a whole new window, most browsers have a way to do this through the preferences settings.

Q: What if I have more than one <a> element with a target? If there's already a "_blank" new window open, will it open in the window that's already open? Or will it open in a new "_blank" window?

A: If you give the name "_blank" to the targets in all your <a> elements, then each link will open in a new blank window. However, this is a good question because it brings up an important point: you don't actually have to name your target "_blank". If you give it another name, say, "coffee", then all links with the target name "coffee" will open in the same window. The reason is that when you give your target a specific name, like "coffee", you are really naming the new window that will be used to display the page at the link. "_blank" is a special case that tells the browser to *always* use a new window.



The Target Attribute Exposed

This week's interview:
Using target considered bad?

Head First: Hello, Target! We're so glad you could join us.

Target: I'm glad to be here. It's nice to know you're still interested in hearing about me.

Head First: Why do you say that?

Target: Well, to be honest, I'm not as popular as I used to be.

Head First: Why do you think that is?

Target: I think it's because users want to be in control of when a window opens. They don't always like new windows popping open at unexpected times.

Head First: Well, it can be very confusing—we've had complaints from people who end up with so many windows on their screens, they can't find the original page.

Target: But it's not like it's difficult to get rid of the windows...just click on the little close button. What's so hard about that?!

Head First: True, but if users don't know a new window has opened, then they can get confused. Sometimes the new window completely covers the old window and it's hard to tell what's happening.

Target: Well, browsers are getting better at this kind of thing.

Head First: How so?

Target: Browsers often open external pages in a new tab, within the same browser window, rather than opening them in a brand-new window.

Head First: Ah, yes, that would help because it will be a lot less confusing to see a new tab open, which the user can visit whenever they want. Unlike opening a new window, it isn't so disorienting.

Head First: How does this help with screen readers though?

Target: You mean browsers used by the visually impaired?

Head First: Right. Some screen readers play a sound when a new window opens, but others just ignore the new window completely, or else they jump right to the new window immediately. Either way, it's gotta be confusing for someone who can't see what's going on. I have no idea how they are handling tabs.

Target: [Sigh] Yeah, we just aren't there yet in terms of providing good tools that meet everyone's needs, especially the visually impaired. That said, we seem to need to have the ability to take the user to pages outside our own site, and many sites do that by opening another window (or tab, if the browser supports it).

Head First: Yup. We need you, but we need to get better about not confusing the user.

Target: I'm hoping the web standard and browser teams will make all this better.

Head First: I guess for now we're just going to have to remember to use you when it's appropriate, but to keep in mind those people who might be visually impaired and not overuse you.

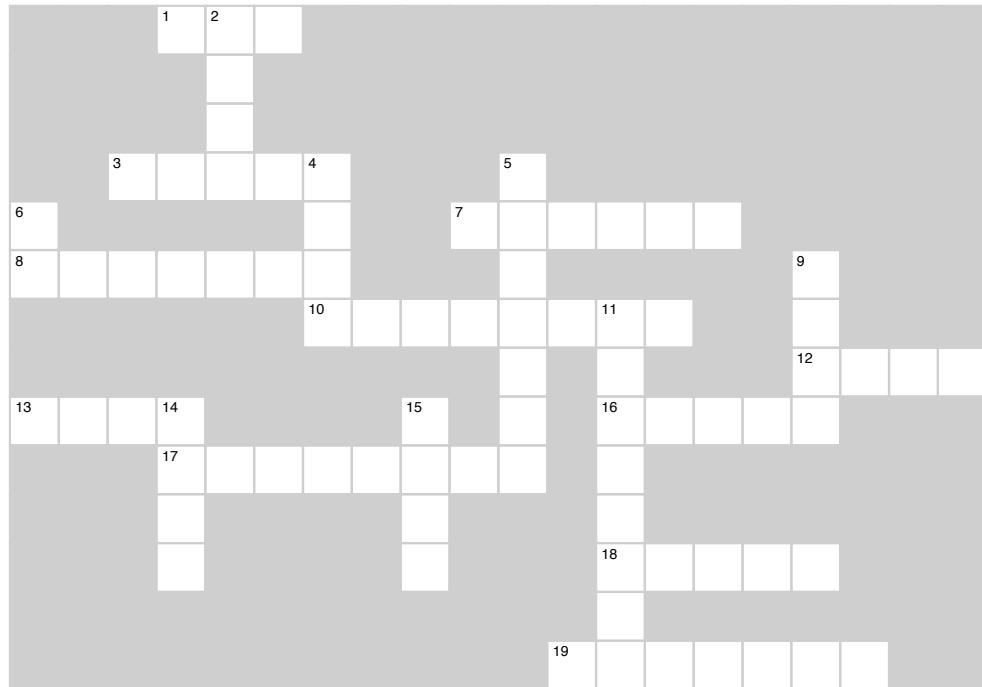
Target: You got it. You've helped ease my burden a bit here; thanks for helping me get the word out!

Head First: Any time, Target!



HTMLcross

Here are some mind benders for your left brain.



Across

1. Web address to a resource.
3. A Mac FTP application.
7. Unique name on the Web.
8. The file you get when you ask for a directory.
10. What are you supposed to send back from Webville?
12. Top directory of your website.
13. Protocol we've been using up until this chapter.
16. People can scan these rather than reading text.
17. Path from the root.
18. Controls domain names.
19. The file you get when you ask for a directory.

Down

2. Top directory of your website.
4. Request/response protocol.
5. Keep your link labels _____.
6. Attribute used to make an element into a destination.
9. Earl sold these.
11. Always use these kinds of links when linking to pages on the same server.
14. Wrong way to pronounce URL.
15. Informative caffeine site.



BULLET POINTS

- Typically the best way to get on the Web is to find a hosting company to host your web pages.
- A domain name is a unique name, like amazon.com or starbuzzcoffee.com, that is used to identify a site.
- A hosting company can create one or more web servers in your domain. Servers are often named “www”.
- The File Transfer Protocol (FTP) is a common means of transferring your web pages and content to a server.
- FTP applications, like Fetch for Mac or WS_FTP for Windows, can make using FTP easier by providing a graphical user interface.
- A URL is a Uniform Resource Locator, or web address, that can be used to identify any resource on the Web.
- A typical URL consists of a protocol, a website name, and an absolute path to the resource.
- HTTP is a request and response protocol used to transfer web pages between a web server and your browser.
- The file protocol is used by the browser to read pages from your computer.
- An absolute path is the path from the root folder to a file.
- “index.html” and “default.htm” are examples of default pages. If you specify a directory without a filename, the web server will look for a default page to return to the browser.
- You can use relative paths or URLs in your <a> element’s href attribute to link to other web pages. For other pages in your site, it’s best to use relative paths, and use URLs for external links.
- Use the id attribute to create a destination in a page. Use # followed by a destination id to link to that location in a page.
- To help accessibility, use the title attribute to provide a description of the link in <a> elements.
- Use the target attribute to open a link in another browser window. Don’t forget that the target attribute can be problematic for users on a variety of devices and alternative browsers.

Wait, wait! Before you go, we need our logo on the web page! Hello? Oh, I guess they’ve already gone on to Chapter 5...

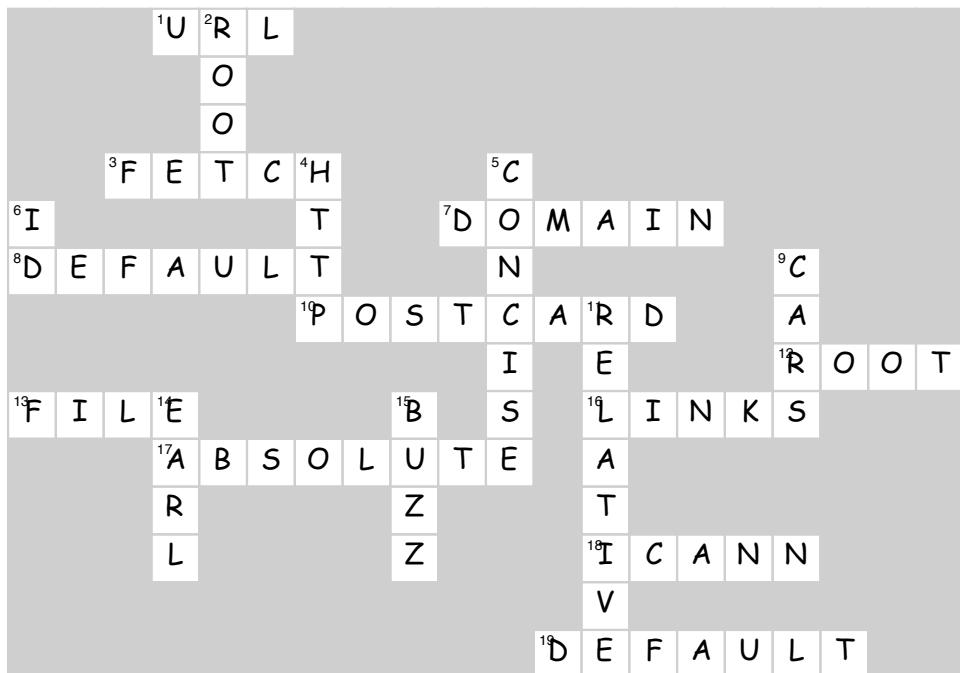




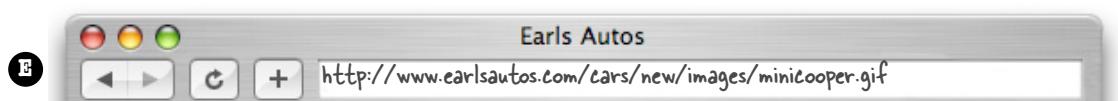
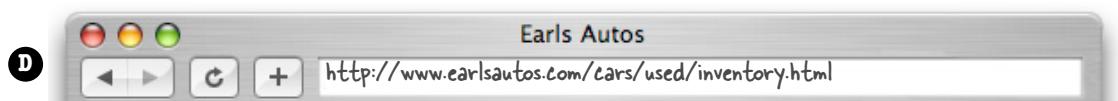
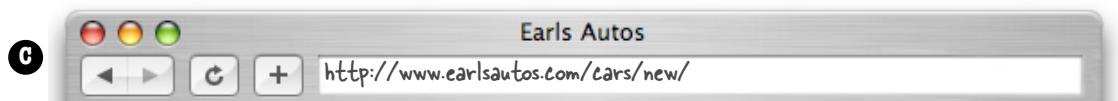
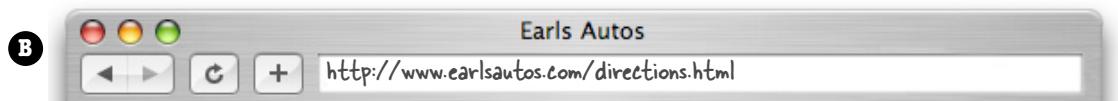
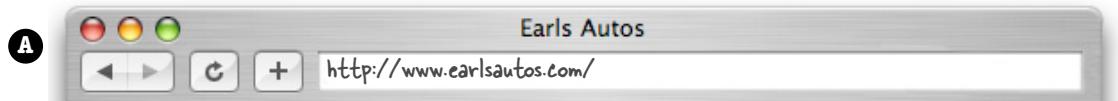
Sharpen your pencil Solution

You've waited long enough. It's time to give your new URL a spin. Before you do, fill in the blanks below and then type in the URL (like you haven't already). If you're having any problems, this is the time to work with your hosting company to get things sorted out. If you haven't set up an hosting company, fill in the blanks for www.starbuzzcoffee.com, and type the URL into your browser anyway.

http :// **website name** /index.html
protocol **absolute path**
 Your website name here.



Earl needs a little help with his URLs





Add a title to the link to "mission.html" with the text "Read more about Starbuzz Coffee's important mission." Notice that we didn't make the mission link's label as concise as it should be. Shorten the link label to "our Mission". Here's the solution; did you test your changes?

```

<html>
  <head>
    <title>Starbuzz Coffee</title>
    <style type="text/css">
      body {
        background-color: #d2b48c;
        margin-left: 20%;
        margin-right: 20%;
        border: 1px dotted gray;
        padding: 10px 10px 10px 10px;
        font-family: sans-serif;
      }
    </style>
  </head>
  <body>
    <h1>Starbuzz Coffee Beverages</h1>
    <h2>House Blend, $1.49</h2>
    <p>A smooth, mild blend of coffees from Mexico,  
Bolivia and Guatemala.</p>

    <h2>Mocha Cafe Latte, $2.35</h2>
    <p>Espresso, steamed milk and chocolate syrup.</p>

    <h2>Cappuccino, $1.89</h2>
    <p>A mixture of espresso, steamed milk and foam.</p>

    <h2>Chai Tea, $1.85</h2>
    <p>A spicy drink made with black tea, spices,  
milk and honey.
    </p>
    <p>
      Read about <a href="mission.html"
      title="Read more about Starbuzz Coffee's important mission">our Mission</a>.
      <br>
      Read the <a href="http://wickedlysmart.com/buzz"
      title="Read all about caffeine on the Buzz">Caffeine Buzz</a>.
    </p>
  </body>
</html>

```

Add a title attribute to the mission link.

Move the "Read about" outside the <a> element.

5 adding images to your pages

Meeting the Media



Smile and say “cheese.” Actually, smile and say “gif,” “jpg,” or “png”—these are going to be your choices when “developing pictures” for the Web. In this chapter you’re going to learn all about adding your first media type to your pages: images. Got some digital photos you need to get online? No problem. Got a logo you need to get on your page? Got it covered. But before we get into all that, don’t you still need to be formally introduced to the `` element? So sorry, we weren’t being rude; we just never saw the “right opening.” To make up for it, here’s an entire chapter devoted to ``. By the end of the chapter you’re going to know all the ins and outs of how to use the `` element and its attributes. You’re also going to see exactly how this little element causes the browser to do extra work to retrieve and display your images.

How the browser works with images

Browsers handle `` elements a little differently than other elements. Take an element like an `<h1>` or a `<p>`. When the browser sees these tags in a page, all it needs to do is display them. Pretty simple. But when a browser sees an `` element, something very different happens: the browser has to retrieve the image before it can be displayed in a page.

The best way to understand this is to look at an example. Let's take a quick look back at the elixirs page from the Head First Lounge, which has four `` elements:

```
<html>
  <head>
    <title>Head First Lounge Elixirs</title>
  </head>
  <body>
    <h1>Our Elixirs</h1>
    <h2>Green Tea Cooler</h2>
    <p>
      
      Chock full of vitamins and minerals, this elixir combines
      the healthful benefits of green tea with a twist of chamomile
      blossoms and ginger root.
    </p>
    <h2>Raspberry Ice Concentration</h2>
    <p>
      
      Combining raspberry juice with lemon grass, citrus peel and
      rosehips, this icy drink will make your mind feel clear and
      crisp.
    </p>
    <h2>Blueberry Bliss Elixir</h2>
    <p>
      
      Blueberries and cherry essence mixed into a base of
      elderflower herb tea will put you in a relaxed state of
      bliss in no time.
    </p>
    <h2>Cranberry Antioxidant Blast</h2>
    <p>
      
      Wake up to the flavors of cranberry and hibiscus in
      this vitamin C rich elixir.
    </p>
    <p>
      <a href="../lounge.html">Back to the Lounge</a>
    </p>
  </body>
</html>
```



We've got four images
in this HTML.

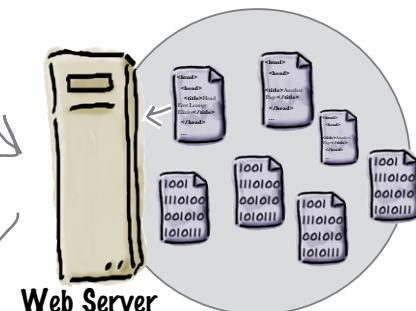
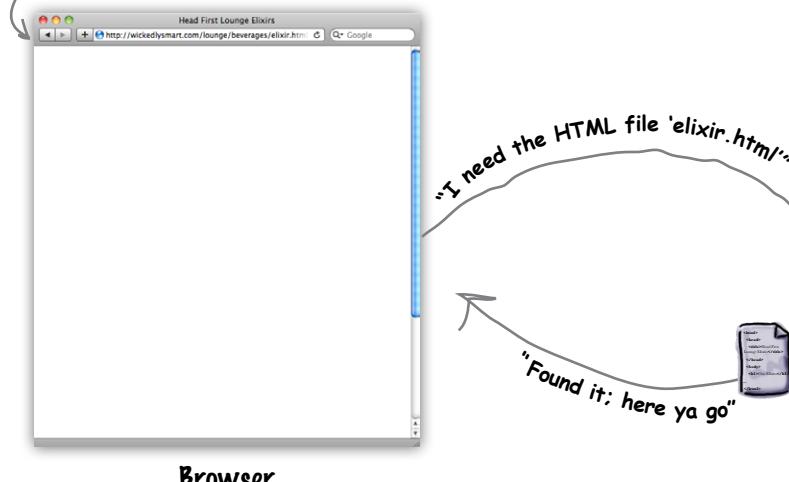
Now let's take a look behind the scenes and step through how the browser retrieves and displays this page when it is requested from <http://wickedlysmart.com/lounge/>:

Behind the Scenes



- ① First, the browser retrieves the file "elixir.html" from the server.

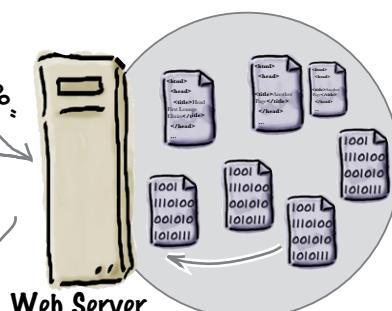
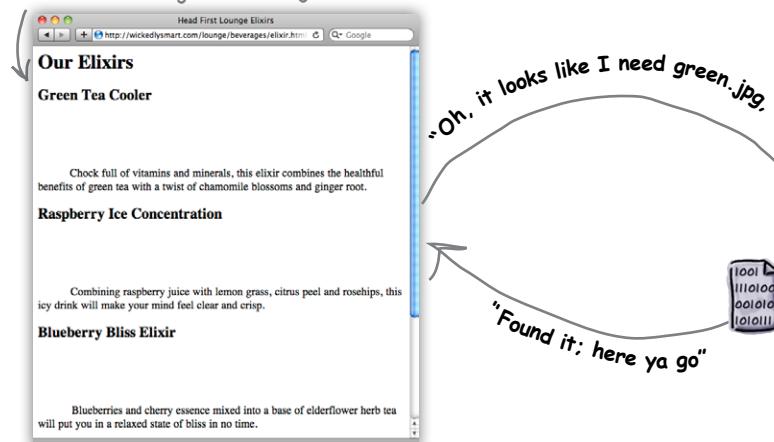
Empty browser window; nothing retrieved yet.



Browser

- ② Next the browser reads the "elixir.html" file, displays it, and sees it has four images to retrieve. So, it needs to get each one from the web server, starting with "green.jpg".

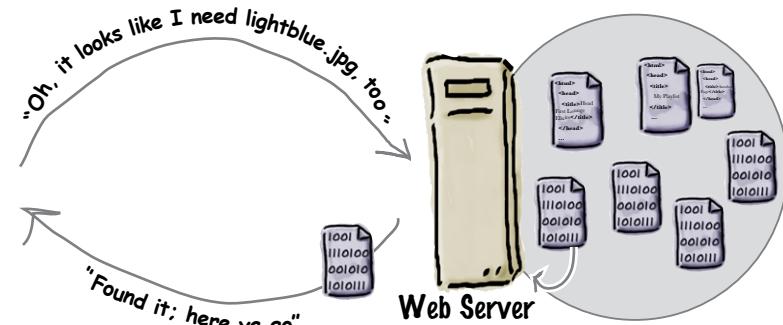
The HTML page is retrieved, but the browser still needs to get the images.



Browser

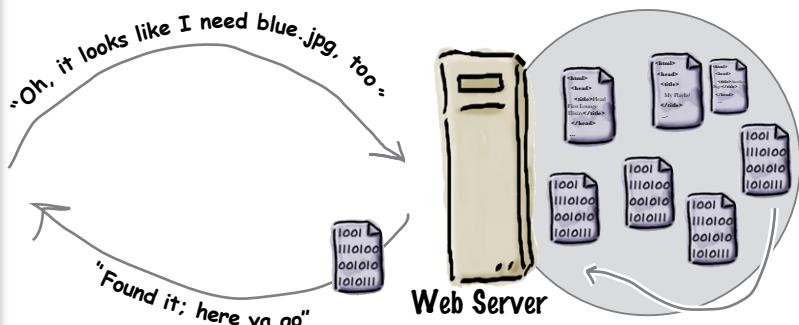
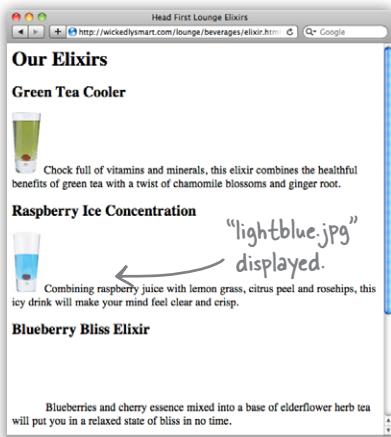
how browsers load images

- ③ Having just retrieved "green.jpg", the browser displays it and then moves on to the next image: "lightblue.jpg".



Browser

- ④ Now the browser has retrieved "lightblue.jpg", so it displays that image and then moves on to the next image, "blue.jpg". This process continues for each image in the page.



Browser

How images work

Images are just images, right? Well, actually there are a zillion formats for images out there in the world, all with their own strengths and weaknesses. But luckily, only three of those formats are commonly used on the Web: JPEG, PNG, and GIF. The only tricky part is deciding which to use when.

So, what are the differences among JPEG, PNG, and GIF?



Use JPEG for photos and complex graphics

Works best for continuous tone images, like photographs.

Can represent images with up to 16 million different colors.

Is a “lossy” format because to reduce the file size, it throws away some information about the image.

Does not support transparency.

Files are smaller for more efficient web pages.

No support for animation.



Use PNG or GIF for images with solid colors, logos, and geometric shapes.

PNG works best for images with a few solid colors, and images with lines, like logos, clip art, and small text in images.

PNG can represent images with millions of different colors. PNG comes in three flavors: PNG-8, PNG-24, and PNG-32, depending on how many colors you need to represent.

PNG compresses the file to reduce its size, but doesn’t throw anything away. So, it is a “lossless” format.

Allows colors to be set to “transparent” so that anything underneath the image will show through.

Files tend to be larger than their JPEG equivalents, but can be smaller or larger than GIF depending on the number of colors used.

Like PNG, GIF works best for images with a few solid colors, and images with lines, like logos, clip art, and small text in images.

GIF can represent images with up to 256 different colors.

GIF is also a “lossless” format.

GIF also supports transparency, but allows only one color to be set to “transparent.”

Files tend to be larger than their JPEG equivalents.

Supports animation.



Would the real image format please stand up?

This week's interview: Image formats mix it up

Head First: Well, hello everyone. I think this might be the first time we've interviewed three interviewees at once!

JPEG: Hey there, and hey to GIF and PNG.

GIF: I'm not sure why I have to share the interview couch with these other bozos. Everyone knows GIF is the original image format of the Web.

JPEG: Ha! As soon as you get good at representing complex images, like photos, maybe then people will take you seriously again, but I'm not sure how you're going to do that with only 256 colors.

Head First: PNG, help us out here? You've been kind of quiet so far...

PNG: Yeah, it's easy to be quiet when you're #1. I can represent complex images like JPEG and I'm also lossless like GIF. Truly the best of both worlds.

Head First: Lossless?

PNG: Right; when you store an image in a lossless format, you don't lose any of the information, or detail, in the image.

GIF: Me too! I'm lossless too, you know.

Head First: Well, why would anyone want a lossy format?

JPEG: There's always a tradeoff. Sometimes what you want is a fairly small file you can download fast, but that has great quality. We don't always need perfect quality. People are very happy with JPEG images.

PNG: Sure, sure, but have you ever looked at lines, logos, small text, solid colors? They don't look so great with JPEG.

Head First: Wait a sec, JPEG raises an interesting issue. So GIF and PNG, are your file sizes large?

PNG: I'll admit my file sizes can be on the large size sometimes, but I provide three formats so you can right-size your images: PNG-8, PNG-24, and PNG-32.

GIF: Sounds like complexity to me—more things for your users to remember.

PNG: Well, GIF, wouldn't the world be nice if we could fit all images into 256 colors? But we can't.

GIF: Hey, for line drawings, figures, that kind of thing, it's often very easy to fit images into 8 bits, and for that I look great.

JPEG: Ha, when is the last time you saw a photo stored in GIF? People have figured out your downsides, GIF.

GIF: Did I mention I can be transparent? You can take parts of me, and anything behind me shows right through.

PNG: You can't compete with me on that one, GIF. I can set any number of colors to transparent; you are limited to one color.

GIF: One color or many, who cares? One is all you need.

PNG: Not if you want to have anti-aliased transparent areas in your image!

GIF: Huh?

PNG: Yeah, you know, because I allow more than one color to be transparent, so you can have nice soft edges around the transparent areas.

Head First: That sounds like a nice feature. Can you do that, JPEG?

JPEG: No, but I'm not too worried about it; there aren't many photos you'd want to do that to. That's for logos.

PNG: Hmm, I'm seeing my transparency used all over the Web.

Head First: Well, I'll have to think twice before doing a three-person interview again, but it sounds to me like GIF and PNG, you're great for logos and text images; JPEG, you're great for photos; and PNG, you come in handy if we want transparency as well as lots of colors. Bye!

PNG, JPEG, GIF: Wait, no, hold on!!!

WHICH IMAGE FORMAT?

Congratulations: you've been elected "Grand Image Format Chooser" of the day. For each image below, choose the format that would best represent it for the Web.

JPEG or PNG or GIF

Uh, I don't mean to be rude, but we're on the eighth page of the IMAGES chapter and you STILL haven't introduced me! JPEG, PNG, GIF, blah, blah, blah... could you get on with it?

And now for the formal introduction: meet the `` element.

We've held off on the introductions long enough. As you can see, there's more to dealing with images than just the HTML markup. Anyway, enough of that for now...it's time to meet the `` element.

Let's start by taking a closer look at the element (although you've probably already picked up on most of how `` works by now):

The `` element is an inline element. It doesn't cause linebreaks to be inserted before or after it.

Here's the `` element.

``

The `src` attribute specifies the location of an image file to be included in the display of the web page.

You already know `` is a void element.

So, is that it? Not quite. There are a couple of attributes you'll want to know about. And of course you'll also want to know how to use the `` element to reference images on the Web that aren't on your own site. But really, you already know the basics of using the `` element.

Let's work through a few of the finer points of using the `` element, and then put all this knowledge to work.

: it's not just relative links anymore

The `src` attribute can be used for more than just relative links; you can also put a URL in your `src` attribute. Images are stored on web servers right alongside HTML pages, so every image on the Web has its own URL, just like web pages do.

You'll generally want to use a URL for an image if you're pointing to an image at a *different* website (remember, for links and images on the *same* site, it's better to use relative paths).

Here's how you link to an image using a URL:

```

```

↑ To include an image using its URL,
just put the whole URL of the
image in the `src` attribute.

The URL is the path to the image,
so the filename at the end is always
the filename of an image. There's
no such thing as a default image like
there is for web pages.



Sharpen your pencil

Here's a "Sharpen your pencil" that is actually about pencils (oh, and images too). This exercise involves a bit of trivia: *Given a typical, brand-new pencil, if you drew one continuous line with it, using the entire pencil up, how long would the line be?*

What's that got to do with images? To find the answer, you're going to have to write some HTML. The answer to this trivia is contained in the image that is at the URL <http://wickedlysmart.com/hfhtmlcss/trivia/pencil.png>. Your job is to add an image to this HTML and retrieve the answer:

```
<html>
  <head>
    <title>Sharpen your pencil trivia</title>
  </head>
  <body>
    <p>How long a line can you draw with the typical pencil?</p>
    <p>
      </p>
    </body>
</html>
```



Put your image element here.

there are no Dumb Questions

Q: So the `` element is quite simple—it just provides a way to specify the location of the image that needs to be displayed in the page?

A: Yes, that about sums it up. We'll talk about a couple of attributes you can add to the element. Later, you'll see a number of ways to use CSS to change the visual style of an image.

But there's a lot to know about the images themselves. What are the different image formats for? When should I use one over the other? How big should they be? How do I prepare images for use in a web page?

Q: We've learned that void elements are elements without content or a closing tag. We've also learned that the `` element is void. But doesn't it have content (the image)?

A: Well, to be more precise, a void element is an element that doesn't have any content in the HTML page to put the opening and closing tags around. True, an image is content, but the `` element refers to the image. The image isn't part of the HTML page itself. Instead, the image replaces the `` element when the browser displays the page. And remember, HTML pages are purely text, so the image could never be directly part of the page. It's always a separate thing.

Q: Back to the example of a web page loading with images...when I load a web page, I don't see the images loading one after the other. Why?

A: Browsers often retrieve the images *concurrently*. That is, the browser makes requests for multiple images at the same time. Given the speed of computers and networks, this all happens fast enough that you usually see a page display along with its images.

Q: If I see an image on a web page, how do I determine its URL so that I can link to it?

A: Most browsers allow you to right-click on an image, which brings up a contextual menu with some choices. In these choices, you should see "Copy Image Address" or "Copy Image Link," which will place the URL in your clipboard. Another way to find the URL is to right-click and choose "Open Image in New Window," which will open the image in a browser window. Then you can get the URL of the image from the browser's address bar. A last option is to use your browser's View Source menu option and look through the HTML. Keep in mind, though, you might find a relative link to the image, so you'll have to "reconstruct" the URL using the website domain name and the path of the image.

Q: What makes a JPEG photo better than a GIF or PNG photo, or a GIF or PNG logo better than a JPEG logo?

A: "Better" is usually defined as some combination of image quality and file size. A JPEG photo will usually be much smaller than an equivalent-quality PNG or GIF, while a PNG or GIF logo will usually look better, and may have a smaller file size than in JPEG format.

Q: How do I choose between GIF and PNG? It seems like they are very similar.

A: PNG is the latest newcomer in graphic formats, and an interesting one because it can support both photos as well as logos. It also has more advanced transparency features than GIF. PNG is supported by all major browsers now, which wasn't true just a few years ago.

To choose between GIF and PNG, there are a few things to consider. First, PNG has slightly better compression than GIF, so for an image with the same number of colors (i.e., up to 256), your PNG file may be smaller. If you need more colors than GIF can offer, and JPEG isn't an option (for instance, you need transparency), PNG is definitely the way to go. However, if you need animation, then you should go with GIF because GIF is the only widely supported format that supports animation.

Always provide an alternative

One thing you can be sure of on the Web is that you never know exactly which browsers and devices will be used to view your pages. Visitors are likely to show up with mobile devices, screen readers for the visually impaired, browsers that are running over very slow Internet connections (and may retrieve only the text, not the images, of a site), cell phones, Internet-enabled T-shirts...Who knows?

But in the middle of all this uncertainty, *you can be prepared*. Even if a browser can't display the images on your page, there is an alternative. You can give the visitor some indication of what information is in the image using the `` element's `alt` attribute. Here's how it works:

```

```

The alt attribute requires a bit of text that describes the image.

If the image can't be displayed, then this text is used in its place. It's like if you were reading the web page over the phone to someone, the alt text is what you'd say in place of the image.



In this exercise you're going to see how your browser handles the `alt` attribute when you have a broken image. The theory goes that if an image can't be found, the `alt` attribute is displayed instead. But not all browsers implement this, so your results may vary. Here's what you need to do:

- 1 Take your HTML from the previous exercise.
- 2 Update the image element to include the `alt` attribute
“The typical new pencil can draw a line 35 miles long.”
- 3 Change the image name of “pencil.png” to “broken.png”. This image doesn't actually exist, so you'll get a broken image.
- 4 Reload the page in your browser.
- 5 Finally, download a couple of other browsers and give this a try. Did you get different results?

Look at the end of the chapter to see our results...

For instance, you could try Firefox (<http://www.mozilla.org/>) or Opera (<http://www.opera.com/>).

Sizing up your images

There's one last attribute of the `` element you should know about—actually, they're a pair of attributes: `width` and `height`. You can use these attributes to tell the browser, up front, the size of an image in your page.

Here's how you use `width` and `height`:

```

```



The `width` attribute tells the browser how wide the image should appear in the page.



The `height` attribute tells the browser how tall the image should appear in the page.

Both `width` and `height` are specified using the number of pixels. If you're not familiar with pixels, we'll go into what they are in a little more detail later in this chapter. You can add `width` and `height` attributes to any image; if you don't, the browser will automatically determine the size of the image before displaying it in the page.

there are no Dumb Questions

Q: Why would I ever use these attributes if the browser just figures it out anyway?

A: On many browsers, if you supply the `width` and `height` in your HTML, then the browser can get a head start laying out the page before displaying it. If you don't, the browser often has to readjust the page layout after it knows the size of an image. Remember, the browser downloads images after it downloads the HTML file and begins to display the page. The browser can't know the size of the images before it downloads them unless you tell it.

You can also supply `width` and `height` values that are larger or smaller than the size of the image and the browser will scale the image

to fit those dimensions. Many people do this when they need to display an existing image at a size that is larger or smaller than its original dimensions. As you'll see later, however, there are many reasons not to use `width` and `height` for this purpose.

Q: Do I have to use these attributes in pairs? Can I just specify a `width` or a `height`?

A: You can, but if you're going to go to the trouble to tell the browser one dimension, supplying the second dimension is about the same amount of work (and there isn't a lot to be gained by supplying just a `width` or a `height` unless you're scaling the image to a particular `width` or `height`).

Q: We've said many times that we are supposed to use HTML for structure, and not for presentation. These feel like presentation attributes. Am I wrong?

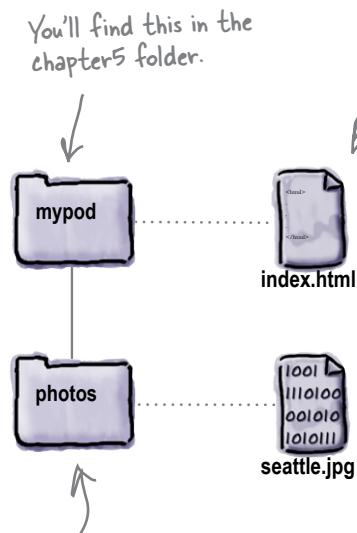
A: It depends on how you are using these attributes. If you're setting the image `width` and `height` to the correct dimensions, then it is really just informational. However, if you are using the `width` and `height` to resize the image in the browser, then you are using these attributes for presentation. In that case, it's probably better to consider using CSS to achieve the same result.

Creating the ultimate fan site: myPod

iPod owners love their iPods, and they take them everywhere. Imagine creating a new site called “myPod” to display pictures of your friends and their iPods from their favorite locations, all around the world.

What do you need to get started? Just some knowledge of HTML, some images, and a love for your iPod.

We've already written some of the HTML for this site, but we haven't added the images yet—that's where you come in. But before you get to the images, let's get things set up; look for the “chapter5” folder in the sample source for the book. There you'll find a folder named “mypad”. Open the “mypad” folder, and here's what you'll see inside:



You'll find this in the chapter5 folder.

We've already written some of the HTML for the myPod site. You'll find it in the “index.html” file.

Here's the first iPod image: an image of Seattle.

We're going to use the photos folder to hold the images for the site.

Note: you'll find a couple of other folders in “mypad” too, but ignore those for now.



iPhones are fine too!

My iPod in Seattle! You can see the Space Needle. You can't see the 628 coffee shops.

Check out myPod's "index.html" file

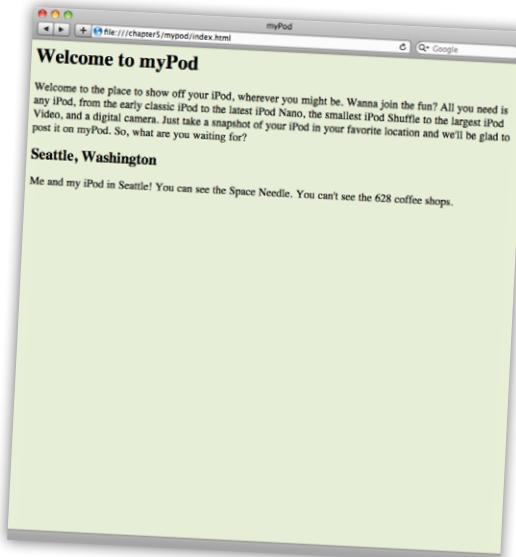
Open up the file "index.html", and you'll see work has already begun on myPod. Here's the HTML so far:

```
<html>
  <head>
    <title>myPod</title>
    <style type="text/css">
      body { background-color: #eaf3da; }
    </style>
  </head>
  <body>
    <h1>Welcome to myPod</h1>
    <p>
      Welcome to the place to show off your iPod, wherever you might be.
      Wanna join the fun? All you need is any iPod from the early classic
      iPod to the latest iPod Nano, the smallest iPod Shuffle to the largest
      iPod Video, and a digital camera. Just take a snapshot of your iPod in
      your favorite location and we'll be glad to post it on myPod. So, what
      are you waiting for?
    </p>
    <h2>Seattle, Washington</h2>
    <p>
      Me and my iPod in Seattle! You can see the
      Space Needle. You can't see the 628 coffee shops.
    </p>
  </body>
</html>
```

And here's how it looks
in the browser. Not bad,
but we need images.

We threw in some **Ready Bake CSS** here.
Just type this in for now. All it does is give
the page a light green background. We'll be
getting to CSS in a few chapters, promise!

This HTML should look familiar, as we're using
the basic building blocks: <h1>, <h2>, and <p>.





Sharpen your pencil

As you can see, a lot of the HTML is already written to get myPod up and running. All you need to do is add an `` element for each photo you want to include. There's one photo so far, "seattle_video.jpg", so go ahead and add an element to place that image in the page below. When you've finished, load the page in your browser and check out the view of Seattle.

```

<html>
  <head>
    <title>myPod</title>
    <style type="text/css">
      body { background-color: #eaf3da; }
    </style>
  </head>
  <body>

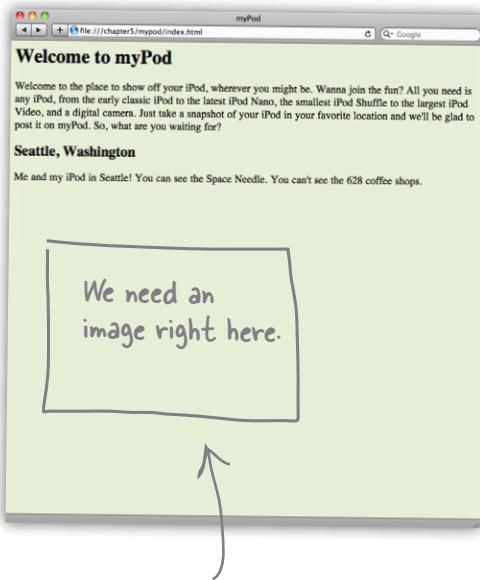
    <h1>Welcome to myPod</h1>
    <p>
      Welcome to the place to show off your iPod, wherever you might be.
      Wanna join the fun? All you need is any iPod from, the early classic
      iPod to the latest iPod Nano, the smallest iPod Shuffle to the largest
      iPod Video, and a digital camera. Just take a snapshot of your iPod in
      your favorite location and we'll be glad to post it on myPod. So, what
      are you waiting for?
    </p>

    <h2>Seattle, Washington</h2>
    <p>
      Me and my iPod in Seattle! You can see the
      Space Needle. You can't see the 628 coffee shops.
    </p>

    <p>
      Your <img> element is
      going to go right here.
    </p>

  </body>
</html>

```



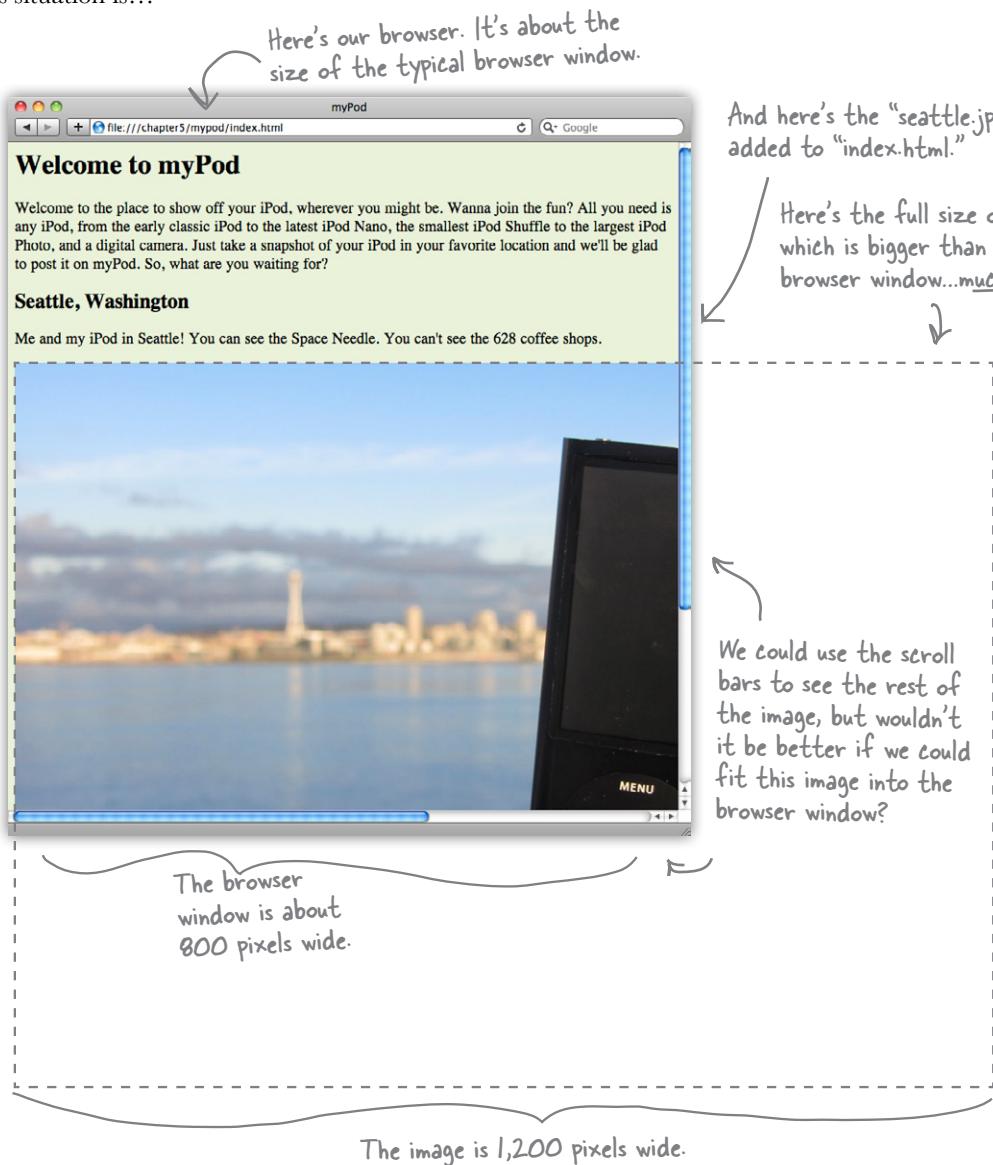
This is where you need to place the first photo.

the image is too big

Whoa! The image is way too large

Well, the image is right there where it should be, but that is one *large* image. Then again, most of the images that come from digital cameras these days are that large (or larger). Should we just leave the image like it is and let visitors use the scroll bar? You're going to see there are a couple of reasons why that's a bad idea.

Let's take a look at the image and the browser and see just how bad this situation is...



Watch it!

If the image fits nicely in your browser window, then your browser may have an "auto image resize" option turned on. More on this in just a sec...

there are no Dumb Questions

Q: What's wrong with having the user just use the scroll bar to see the image?

A: In general, web pages with large images are hard to use. Not only can your visitors not see the entire image at once, but also using scroll bars is cumbersome. Large images also require more data to be transferred between the server and your browser, which takes a lot of time and may result in your page being very slow to display, particularly for users on dial-up or other slow connections.

Q: Why can't I just use the width and height attributes to resize the images on the page?

A: Because the browser still has to retrieve the entire large image before it scales it down to fit your page.

Q: You said the browser window is 800 pixels wide; what exactly does that mean?

A: Your computer's display is made up of millions of dots called pixels. If you look very closely at your display, you'll see them:



And while screen sizes and resolutions tend to vary (some people have small monitors, some large), most people typically set their browsers to somewhere between 800 and 1,280 pixels wide. So, around 800 pixels is a good rule of thumb for the maximum width of your images (and your web pages too, but we'll get to that in a later chapter).

Q: How do the number of pixels relate to the size of the image on the screen?

A: A good rule of thumb is 96 pixels to every inch, although with today's high resolution monitors and retinal displays, it can go higher. We used to use a standard of 72 pixels per inch (ppi), but to handle modern displays, the concept of a CSS pixel has been created. The CSS pixel is 1/96 of an inch (96 ppi). So for a 3" wide × 3" high image, you'd use 96 (pixels) × 3 (inches) = 288 × 288 pixels.

Q: Well, how large should I make my images then?

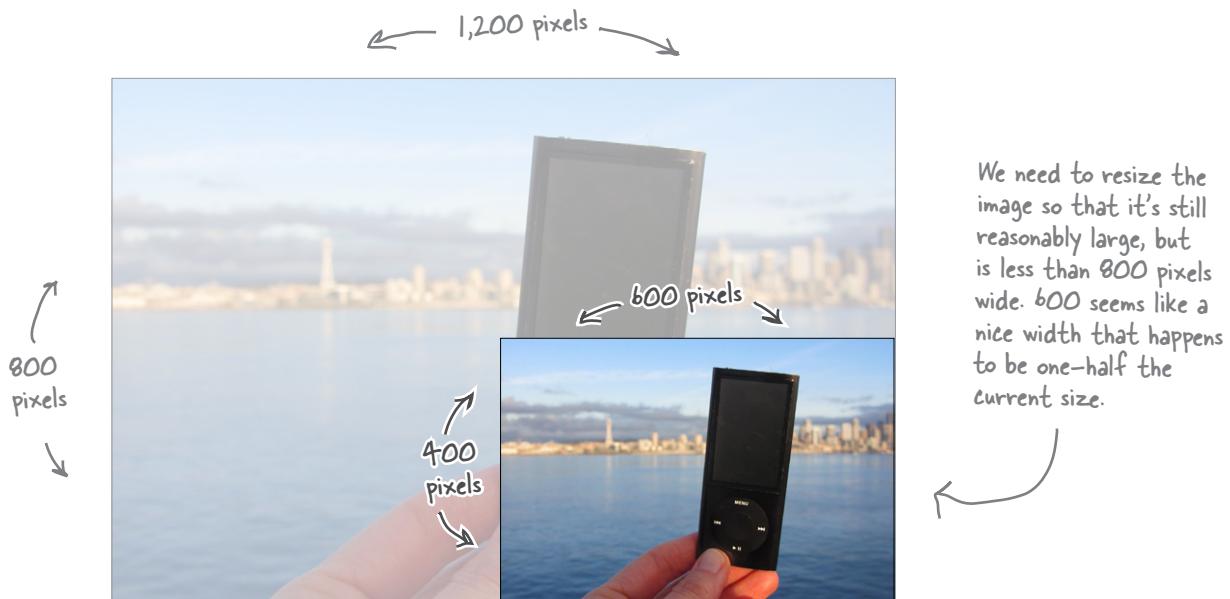
A: In general, you want to keep the width of your image to less than 800 pixels wide. Of course, you may want your images to be significantly smaller (or somewhat larger) depending on what you're using the image for. What if the image is a logo for your page? You probably want that small, but still readable. After all, you don't need a logo the width of the entire web page. Logos tend to run between 100 and 200 pixels wide. So, ultimately, the answer to your question depends on the design of your page. For photos—which you usually do want to view as large as possible—you may want to have a page of small thumbnail images that load quickly, and then allow the user to click on each thumbnail to see a larger version of the image. We'll get to all that shortly.

Q: I think my browser automatically resized the image of Seattle, because it fits perfectly in the window. Why did my browser do this?

A: Some browsers have a feature that resizes any image that doesn't fit within the width of your browser. But many browsers don't do this, so you don't want to rely on it. Even if every browser did have this feature, you'd still be transferring a lot more data than necessary between the server and browser, which would make your page slow to load and less usable. And keep in mind that an increasing number of people are viewing web pages on mobile devices, and large images will impact data usage on these devices.

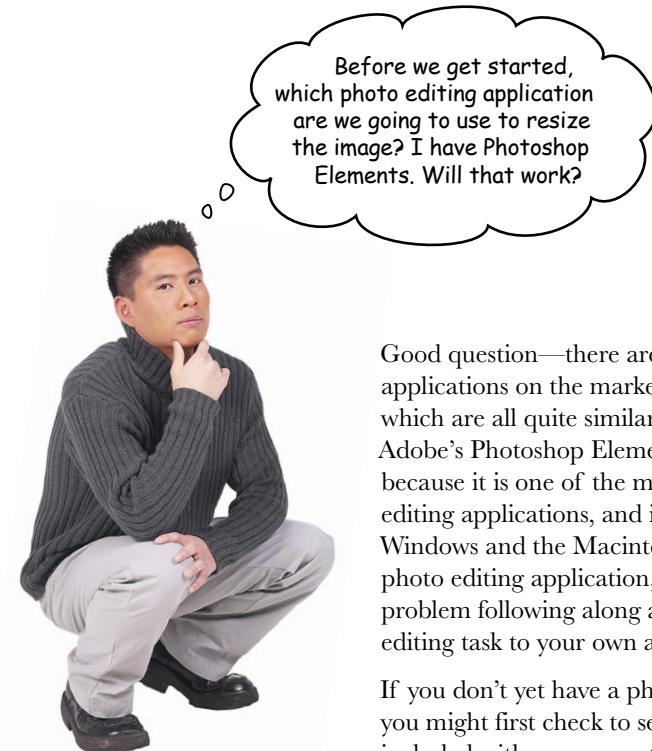
Resize the image to fit in the browser

Let's fix up this image so it fits the browser page better. Right now, this image is 1,200 pixels wide by 800 pixels tall (you'll see how to determine that in a sec). Because we want the width of the image to be less than 800 pixels, we need to decide on a width that would fit our myPod web page nicely. The whole point of myPod is viewing photos of iPods in their surroundings, so we probably want to have reasonably large images. If we reduce this image size by one-half to 600 pixels wide by 400 pixels high, that will still take up most of the browser width, while still allowing for a little space on the sides. Sound good? Let's get this image resized...



Here's what you're going to do:

- 1 Open the image using a photo editing application.**
- 2 Reduce the image size by one-half (to 600 pixels by 400 pixels).**
- 3 Save the image as “seattle_video_med.jpg”.**



Before we get started,
which photo editing application
are we going to use to resize
the image? I have Photoshop
Elements. Will that work?

Good question—there are lots of photo editing applications on the market (some freely available), which are all quite similar. We’re going to use Adobe’s Photoshop Elements to resize the images, because it is one of the most popular photo editing applications, and is available on both Windows and the Macintosh. If you own another photo editing application, you should have no problem following along and translating each editing task to your own application.

If you don’t yet have a photo editing application, you might first check to see if there was one included with your operating system. If you have a Mac, you can use iPhoto to edit your photos. If you’re a Windows user, you might find Microsoft’s Digital Image Suite on your computer already. If you still don’t have an editing application available to you, follow along and for each step, you can use the HTML and images included in the example folders.

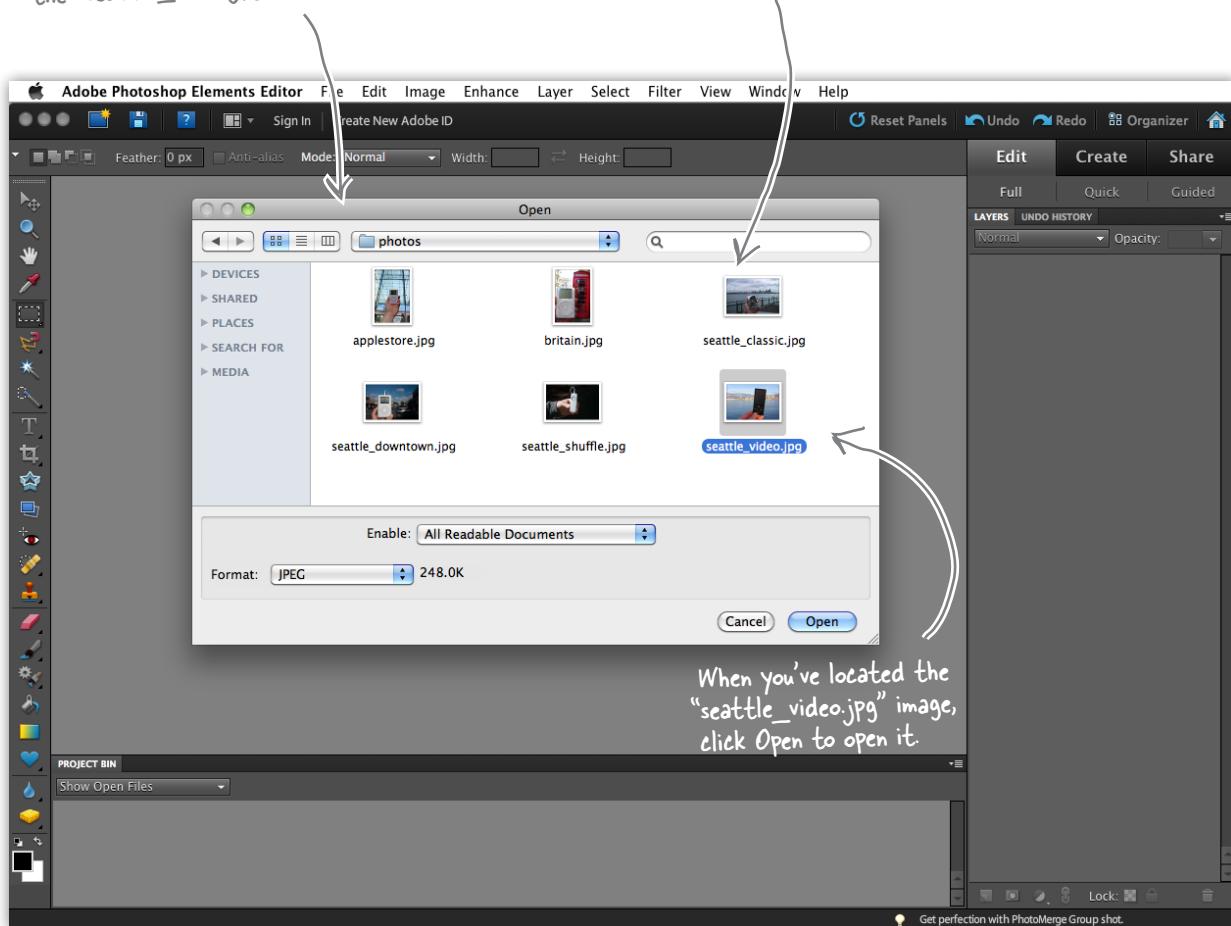
If you don’t have Adobe Photoshop Elements, but
you’d like to follow along for the rest of the chapter
with it, you can download it and try it out free for
30 days. The URL to download it is: http://www.adobe.com/go/tryphotoshop_elements.

Open the image

First, start your photo editing application and open the “seattle_video.jpg” image. In Photoshop Elements, you’ll want to choose the “Open...” menu option under the File menu, which will open the Open dialog box. You’re going to use this to navigate to the image “seattle_video.jpg” in the “chapter5/mypod/photos” folder.

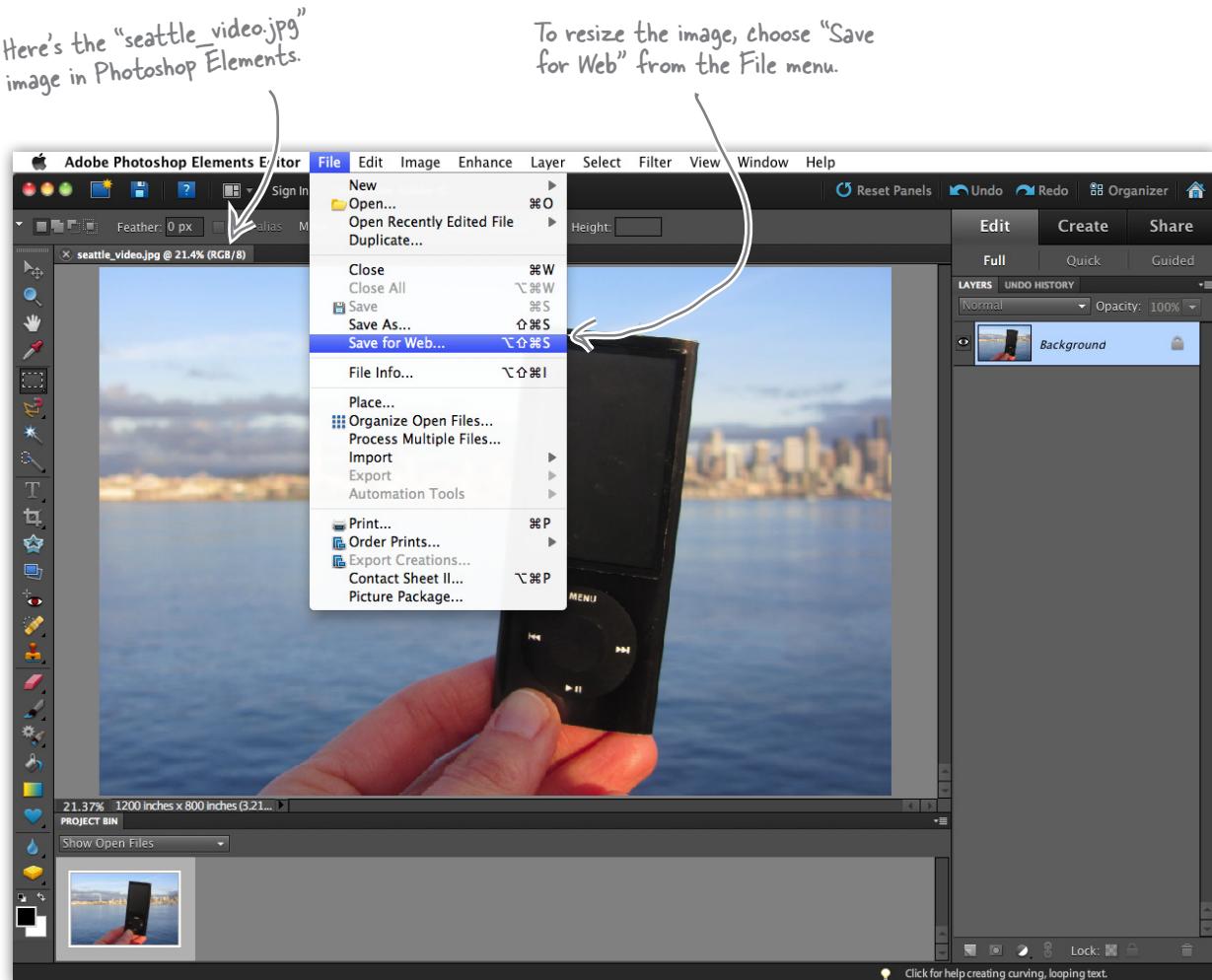
Here's the Open dialog box.
Use this dialog to navigate to
the “seattle_video.jpg” image.

As you navigate through folders, you'll see a
preview of the images in those folders here.



Resizing the image

Now that “seattle_video.jpg” is open, we’re going to use the “Save for Web” dialog to both resize the image and save it. To get to that dialog box, just choose the “Save for Web” menu option from the File menu.

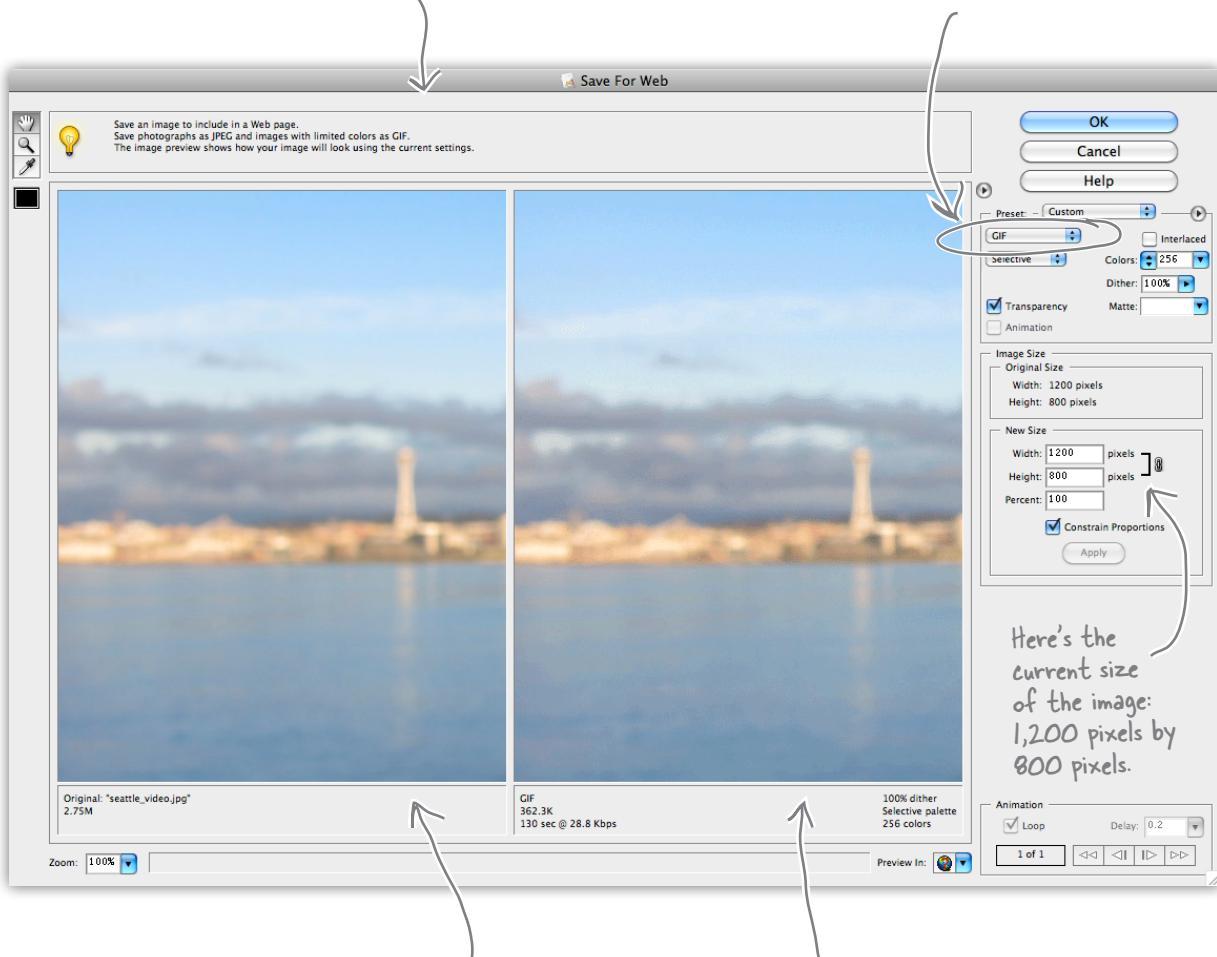


Resizing the image, continued...

After you've selected the "Save for Web" menu option, you should see the dialog box below; let's get acquainted with it before we use it.

This dialog lets you do all kinds of interesting things. For now, we're going to focus on how to use it to resize and save images in JPEG format for web pages.

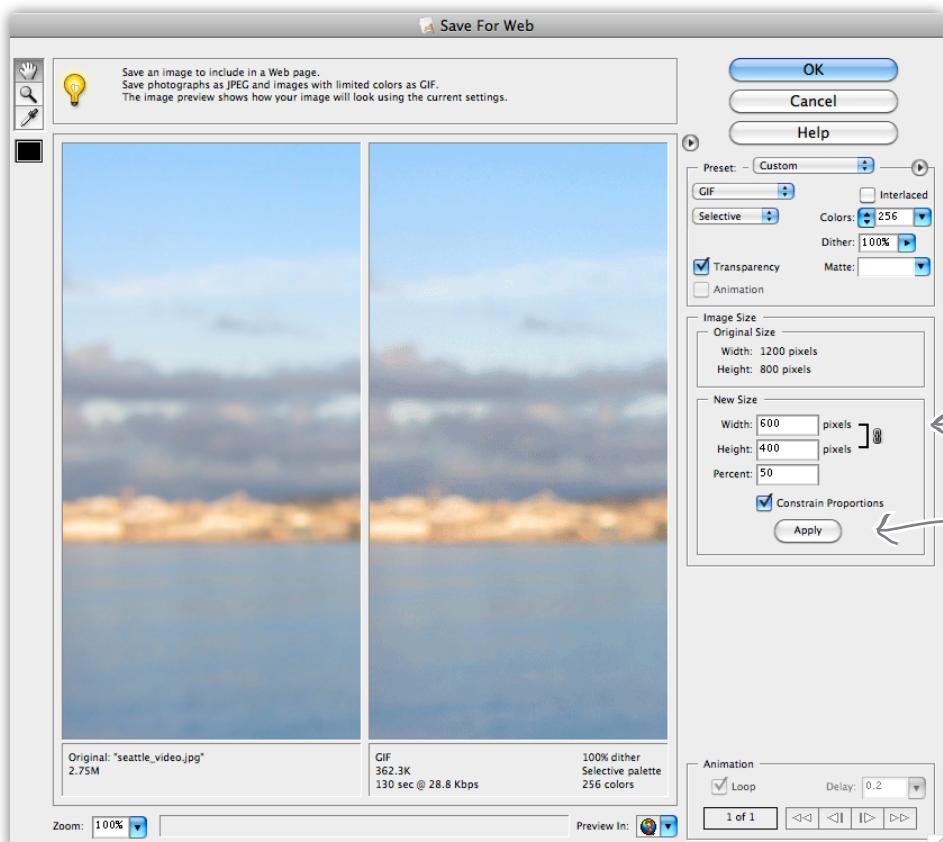
Here's where you choose the format to save your file. Currently it's set to save as GIF; we're going to change this to JPEG in a couple of pages...



This split window shows you your original image on the left, and the image in the format you're saving it for the Web on the right. Currently this is showing GIF format; we'll be changing this to JPEG in an upcoming step.

As you can see, there's a lot of functionality built into this dialog. Let's put it to good use. To resize the image, you need to change the width to 600 pixels and the height to 400 pixels. Then you need to save the image in JPEG format. Let's start with the resize...

(1) Change the image size here to a width of 600 and a height of 400. If you have Constrain Proportions checked, then all you have to do is type the new width, 600, and Elements will change the height to 400 for you.



(2) Once the width and height are set correctly, click Apply to let Elements know this is the size you want.

This will not affect the original image, just the file you're going to save.

You must click Apply to reduce the image size; otherwise, the image will be saved at its original width and height.

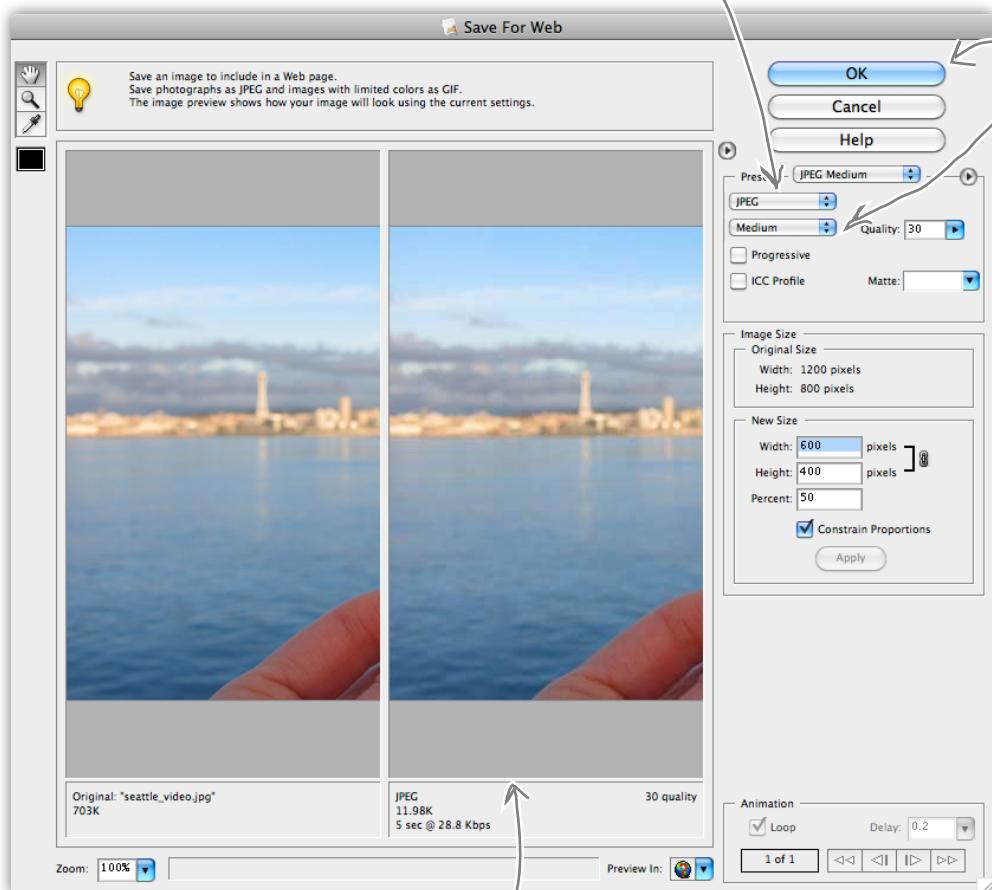
You've resized—now save

Now you just need to save the image in the correct format (JPEG). To do that, you need to choose JPEG format and set the quality to Medium. We'll talk more about quality in a sec.

(1) Now that the image size is set, you just need to choose the format for the image. Currently it's set to save as GIF; change this to JPEG like we've done here.

(2) Set the quality to Medium.

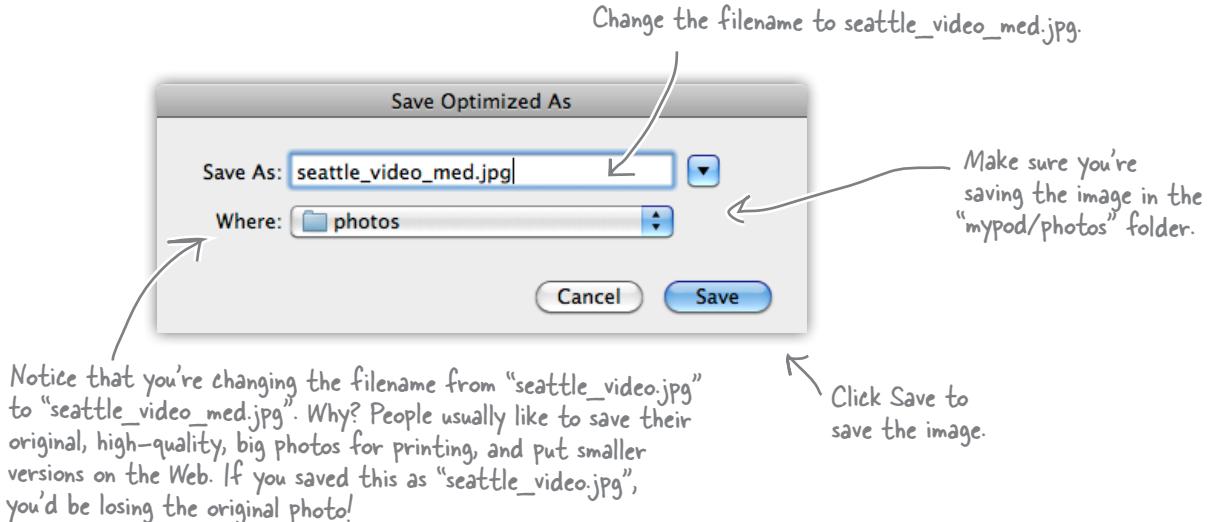
(3) That's it; click OK and go to the next page.



Notice that when you clicked Apply in the previous step, the image was resized and redisplayed.

Save the image

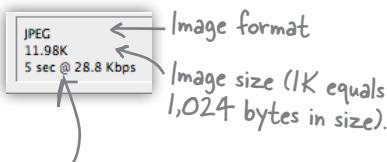
After you click OK, you'll get a Save dialog. Save the image as "seattle_video_med.jpg" so you don't overwrite the original photo.



there are no Dumb Questions

Q: Can you say more about the quality setting in "Save for Web"?

A: The JPEG format allows you to specify the level of image quality you need. The lower the quality, the smaller the file size. If you look at the preview pane in the "Save for Web" dialog, you can see both the quality and file size change as you change the quality settings.



Photoshop Elements even tells you how long it would take to transfer over a dial-up modem to a browser.

The best way to get a feel for quality settings and the various image formats is to experiment with them on your own images. You'll soon figure out what quality levels are needed for your image and the type of web page you're developing. You'll also get to know when to use JPEG versus other formats.

Q: What is the number 30 next to the Quality label in the JPEG Options dialog box?

A: The number 30 is what Photoshop Elements considers Medium quality. JPEG actually uses a scale of 1–100%, and Low, Medium, High, etc. are just preset values that many photo editing applications use.

Q: Couldn't I just use the `` element's width and height attributes to resize my image instead?

A: You could use the width and height attributes to resize an image, but that's not a good idea. Why? Because if you do that, you're still downloading the full-size image, and making the browser do the work to resize the image (just like when you have the auto resize option on in browsers that support that feature). The width and height attributes are really there to help the browser figure out how much space to reserve for the image; if you use them, they should match the actual width and height of the image.

Fixing up the myPod HTML

Once you've saved the image, you can close Photoshop Elements. Now all you need to do is change the myPod "index.html" page to include the new version of the photo, "seattle_video_med.jpg". Here's a snippet of the "index.html" file, showing only the parts you need to change.

```
<html>
  <head>
    <title>myPod</title>
    <style type="text/css">
      body { background-color: #eaf3da; }
    </style>
  </head>
  <body>
    .
    .
    .
    <h2>Seattle, Washington</h2>
    <p>
      Me and my iPod in Seattle! You can see the
      Space Needle. You can't see the 628 coffee shops.
    </p>
    <p>
      
    </p>
  </body>
</html>
```

The rest of the HTML goes here. You've already got it in your "index.html" file.

All you need to do is change the filename in the `` element to the name of the image you just made: "seattle_video_med.jpg".

And now for the test drive...

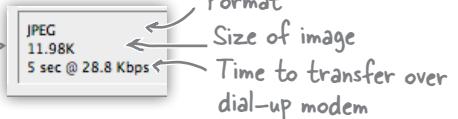
Go ahead and make the changes, save them, and reload "index.html" in your browser. Things should look much better. Now the image is sized just right to give your visitors a good view—without overwhelming them with a large photo.

Now the image fits nicely in the browser window. And it's a smaller file size too, which will help the page load more quickly.



WHICH IMAGE FORMAT?

Your task this time: open the file “chapter5/testimage/eye.jpg” in Photoshop Elements. Open the “Save for Web” dialog and fill in the blanks below by choosing each quality setting for JPEG (Low, Medium, High, etc.), as well as PNG-24 and GIF. You’ll find this information in the preview pane below the image. Once you’ve finished, determine which setting makes the most sense for this image.



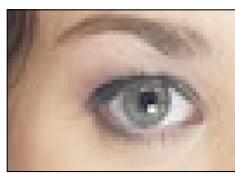
Try PNG-8 too!



Format	Quality	Size	Time	Winner
--------	---------	------	------	--------



<u>PNG-24</u>	<u>N/A</u>	_____	_____	<input type="checkbox"/>
---------------	------------	-------	-------	--------------------------



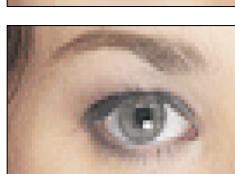
<u>JPEG</u>	<u>Maximum</u>	_____	_____	<input type="checkbox"/>
-------------	----------------	-------	-------	--------------------------



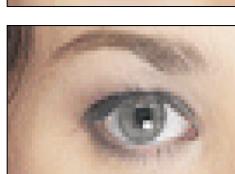
<u>JPEG</u>	<u>High</u>	_____	_____	<input type="checkbox"/>
-------------	-------------	-------	-------	--------------------------



<u>JPEG</u>	<u>Medium</u>	_____	_____	<input type="checkbox"/>
-------------	---------------	-------	-------	--------------------------



<u>JPEG</u>	<u>Low</u>	_____	_____	<input type="checkbox"/>
-------------	------------	-------	-------	--------------------------



<u>GIF</u>	<u>N/A</u>	_____	_____	<input type="checkbox"/>
------------	------------	-------	-------	--------------------------

More photos for myPod

A new batch of photos has arrived for myPod: three more from Seattle and a few from a friend in Britain. The photos have already been resized to less than 800 pixels wide. Add the `` elements for them (you'll find the images already in the photos folder):

```
<html>
  <head>
    <title>myPod</title>
    <style type="text/css">
      body { background-color: #eaf3da; }
    </style>
  </head>
  <body>
    <h1>Welcome to myPod</h1>
    <p>
      Welcome to the place to show off your iPod, wherever you might be.
      Wanna join the fun? All you need is any iPod, from the early classic
      iPod to the latest iPod Nano, the smallest iPod Shuffle to the largest
      iPod Video, and a digital camera. Just take a snapshot of your iPod in
      your favorite location and we'll be glad to post it on myPod. So, what
      are you waiting for?
    </p>

    <h2>Seattle, Washington</h2>
    <p>
      Me and my iPod in Seattle! You can see the
      Space Needle. You can't see the 628 coffee shops.
    </p>

    <p>
      
      
      
      
    </p>

    <h2>Birmingham, England</h2>
    <p>
      Here are some iPod photos around Birmingham. We've obviously got some
      passionate folks over here who love their iPods. Check out the classic
      red British telephone box!
    </p>

    <p>
      
      
    </p>
  </body>
</html>
```

Feel free to add some of your own photos here as well. Just remember to resize them first.

Let's keep all the Seattle photos together.

Same with the Birmingham photos...

Taking myPod for another test drive

At this point we don't need to tell you to reload the page in your browser; we're sure you're way ahead of us. Wow, what a difference a few images make, don't you think? This page is starting to look downright interesting.

But that doesn't mean you're there yet. While you've got a great set of images on the page, and even though you've already resized them, the images are still quite large. Not only is the page going to load more and more slowly as you add more images, but also the user has to do a lot of scrolling to see them all. Wouldn't it be better if users could see a small "thumbnail" image for each photo, and then click on the thumbnail to see the larger image?

And here's what the page looks like now, close up.



Here's what
the whole page
looks like now,
with all the
images.



Reworking the site to use thumbnails

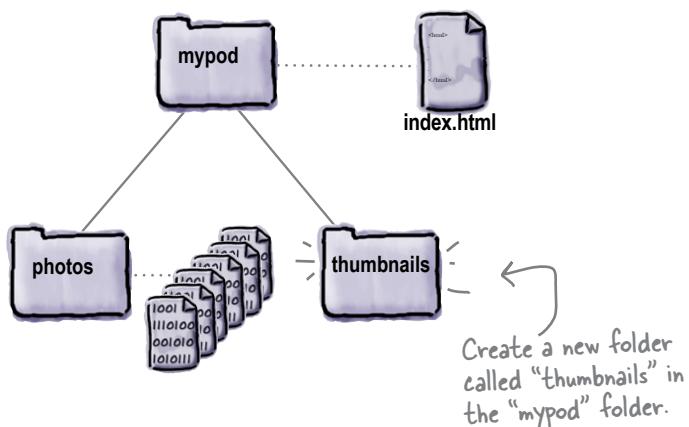
You're now going to make this page more usable by substituting a smaller image (which we call a *thumbnail*) for each photo, and then you'll create a link from that thumbnail to each of the larger photos. Here's how you're going to do this, one step at a time:

- ❶ Create a new directory for the thumbnails.
- ❷ Resize each photo to 150 by 100 pixels and save it in a “thumbnail” folder.
- ❸ Set the `src` of each `` element in “index.html” to the thumbnail version of the photo.
- ❹ Add a link from each thumbnail to a new page containing the larger photo.

Create a new directory for thumbnails

To keep things organized, create a separate folder for the thumbnail images. Otherwise, you'll end up with a folder of larger images and small thumbnails all lumped together, which could get quite cluttered after you've added a significant number of photos.

Create a folder called “thumbnails” under the “mypod” folder. If you're working from the book example files, you'll find this folder already in there.



Create the thumbnails

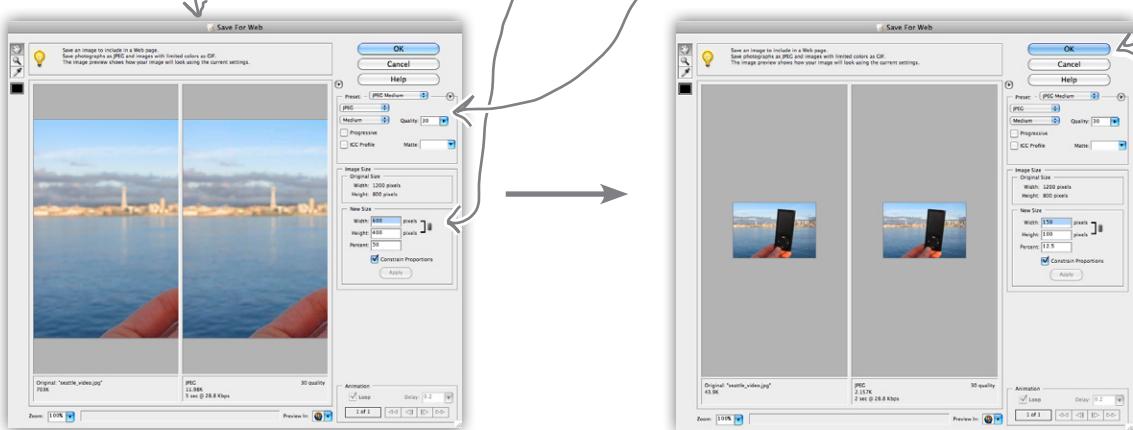
You've got a place to put your thumbnails, so let's create them. Start by opening "seattle_video_med.jpg" with your photo editing application. You're going to resize it to 150 by 100 pixels using the same method you used to create the 600 by 400 version:

In Photoshop Elements,
choose the "Save for Web"
menu option.

Then change the width
to 150 and the height to
100 and click Apply.

Don't forget to change the
format to JPEG, Medium quality.

Finally,
click OK.



With the image resized, choose OK and save it as the same name but *in the thumbnail folder*. **Be careful:** if you save it to the "photos" folder, you'll be replacing the larger image.

Now, repeat this for each photo in your "photos" folder.

If you're working with the example files, you'll find the thumbnails already in the "thumbnails" folder, so you don't have to do every one (after all, you're learning HTML, not batch photo processing).



What about the photos
from Birmingham—they are
taller than they are wide. Does
150x100 make sense?

Good catch. Because these images are taller than they are wide, we have two choices: we can switch the dimensions and make them 100 by 150, or we can crop each image and make a 150-by-100-pixel thumbnail from it. We're going to make ours 100 by 150; feel free to crop them and create 150-by-100-pixel images if you'd like to explore how to do that in your photo editing application.



Rework the HTML to use the thumbnails

Now you just need to change the HTML so that the `` elements get their images from the “thumbnails” folder rather than the “photos” folder. And because you’re currently using relative paths like “photos/seattle_video_med.jpg”, that’s going to be simple: for each `` element, all you need to do is change the folder from “photos” to “thumbnails”.

```
<html>
  <head>
    <title>myPod</title>
    <style type="text/css">
      body { background-color: #eaf3da; }
    </style>
  </head>
  <body>
    <h1>Welcome to myPod</h1>
    <p>
      Welcome to the place to show off your iPod, wherever you might be.
      Wanna join the fun? All you need is any iPod, from the early classic
      iPod to the latest iPod Nano, the smallest iPod Shuffle to the largest
      iPod Video, and a digital camera. Just take a snapshot of your iPod in
      your favorite location and we'll be glad to post it on myPod. So, what
      are you waiting for?
    </p>
    <h2>Seattle, Washington</h2>
    <p>
      Me and my iPod in Seattle! You can see the
      Space Needle. You can't see the 628 coffee shops.
    </p>
    <p>
      
      
      
      
    </p>
    <h2>Birmingham, England</h2>
    <p>
      Here are some iPod photos around Birmingham. We've obviously got some
      passionate folks over here who love their iPods. Check out the classic
      red British telephone box!
    </p>
    <p>
      
      
    </p>
  </body>
</html>
```

All you need to do is change the folder from “photos” to “thumbnails”.

Take myPod for another test drive

Ahhh...much better. Visitors can see all the available pictures at a glance. They can also tell which photos go with each city more easily. Now we need to find a way to link from each thumbnail to the corresponding large image.



Welcome to myPod

Welcome to the place to show off your iPod, wherever you might be. Wanna join the fun? All you need is any iPod, from the early classic iPod to the latest iPod Nano, the smallest iPod Shuffle to the largest iPod Video, and a digital camera. Just take a snapshot of your iPod in your favorite location and we'll be glad to post it on myPod. So, what are you waiting for?

Seattle, Washington

Me and my iPod in Seattle! You can see the Space Needle. You can't see the 628 coffee shops.

Birmingham, England

Here are some iPod photos around Birmingham. We've obviously got some passionate folks over here who love their iPods. Check out the classic red British telephone box!

Wait a sec, don't
you think you're pulling a
fast one? The images used to
be on top of each other; now
they're side by side.



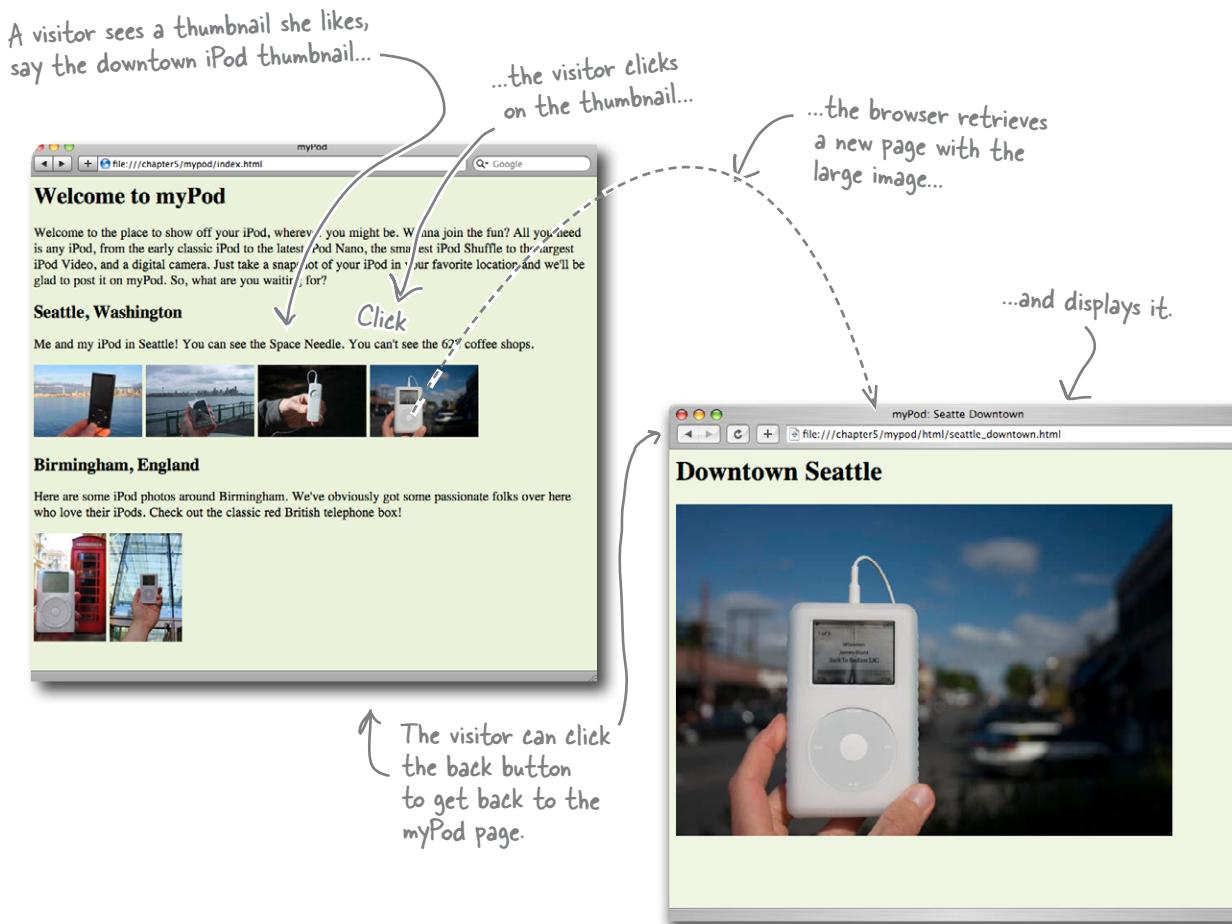
Right, but remember the `` element is an inline element.

In other words, we didn't "pull anything." Because `` is displayed as an inline element, it doesn't cause linebreaks to be inserted before and after the element is displayed. So, if there are several images together in your HTML, the browser will fit them side by side if the browser window is wide enough.

The reason the larger photos weren't side by side is because the browser didn't have room to display them next to each other. Instead, it displayed them on top of each other. A browser always displays vertical space before and after a block element, and if you look back at the screenshots, you'll see the images are right on top of each other with no space in between. That's another sign `` is an inline element.

Turning the thumbnails into links

You're almost there. Now you just need to create a link from each thumbnail image to its larger version. Here's how this is going to work:



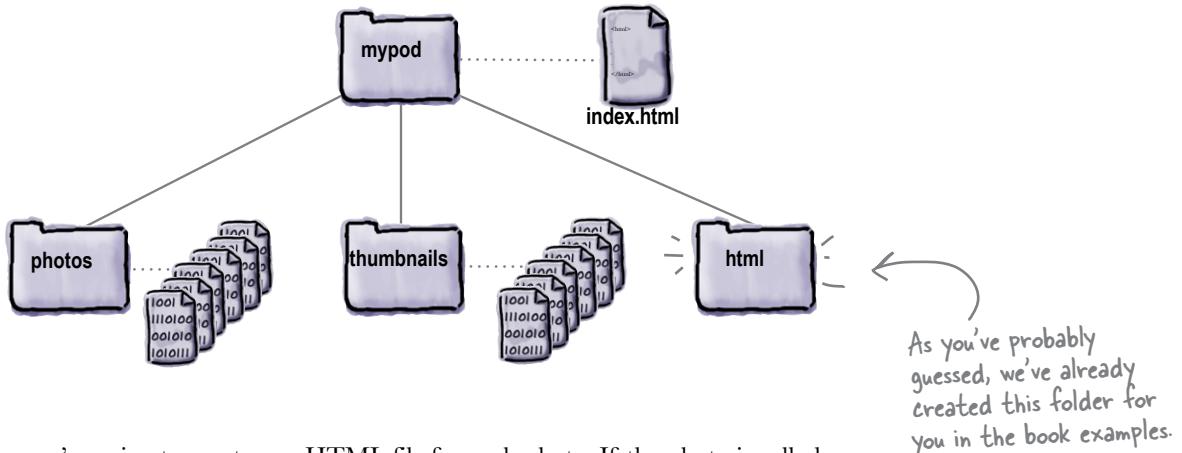
To do this you need two things:

- 1 A page to display each photo along with a heading that describes its content.**
- 2 A link from each thumbnail in “index.html” to its corresponding photo page.**

Let's create the pages first, and then we'll come back and finish off the links.

Create individual pages for the photos

First, create a new folder called “html” to hold these individual pages, just below the “mypod” folder:



Now we’re going to create one HTML file for each photo. If the photo is called “seattle_video_med.jpg”, then let’s call the HTML file “seattle_video_med.html” just to be consistent. In each HTML file, we’ll have a heading that describes the photo, followed by the photo. Here’s the HTML for the first Seattle photo. All the other pages will follow this same format:

```

<html>
  <head>
    <title>myPod: Seattle Ferry</title>
    <style type="text/css"> body { background-color: #eaf3da; } </style>
  </head>
  <body>
    <h1>Seattle Ferry</h1>
    <p>
      
    </p>
  </body>
</html>

```

Title for the page. This should describe the photo.

Here we give the page a descriptive heading.

Here's the `` element that points to the large “seattle_video_med.jpg” photo. Let's also give the image a descriptive alt attribute.

Notice that we need to use “..” in the relative path because the “html” folder is a sibling of the “photos” folder, so we have to go up one folder and then down into “photos” when using relative links.



If you look in the “html” folder in the chapter example files, you’ll find all of the single photo pages already there, except one—the page for “seattle_downtown.jpg”. Create a page called “seattle_downtown.html” in the “html” folder, and test it out. Get this working before you move on. You’ll find the answer in the back of the chapter if you have any problems.

So, how do I make links out of images?

You’ve got your large photos, your smaller thumbnails, and even a set of HTML pages for displaying individual photos. Now you need to put it all together and get those thumbnails in “index.html” linked to the pages in the “html” folder. But how?

To link an image, you put the `` element inside an `<a>` element, like this:

```
Here's the <img> element for  
the "seattle_downtown.jpg"  
thumbnail, just as it is in the  
"index.html" file.  
  
And here's an <a>  
opening tag just before  
the <img> element.  
  
<a href="html/seattle_downtown.html">  
    
  </a>  
  
Here's the  
closing <a> tag.  
  
The <img> element is nested  
directly inside the <a> element.  
  
The href is linked to the new  
HTML page for the photo,  
"seattle_downtown.html", which  
is in the "html" directory.
```

Once you’ve placed the `` element into an `<a>` element, the browser treats the image as a clickable link. When you click the image, the browser will retrieve the page in the `href`.

Add the image links to “index.html”

This is the last step. You just need to wrap `<a>` elements around each thumbnail’s `` element in your “index.html” file. Remember, the href of each `<a>` element should link to the page containing the large version of the image in the “html” folder. Make sure that your links, thumbnails, and pages all match up correctly.

Here’s the complete “index.html” file. All you need to do is add the HTML marked in gray.

```

<html>
  <head>
    <title>myPod</title>
    <style type="text/css">
      body { background-color: #eaf3da; }
    </style>
  </head>
  <body>

    <h1>Welcome to myPod</h1>
    <p>
      Welcome to the place to show off your iPod, wherever you might be.
      Wanna join the fun? All you need is any iPod, from the early classic
      iPod to the latest iPod Nano, the smallest iPod Shuffle to the largest
      iPod Video, and a digital camera. Just take a snapshot of your iPod in
      your favorite location and we'll be glad to post it on myPod. So, what
      are you waiting for?
    </p>

    <h2>Seattle, Washington</h2>
    <p>
      Me and my iPod in Seattle! You can see the
      Space Needle. You can't see the 628 coffee shops.
    </p>

    <p>
      <a href="html/seattle_video_med.html">
        
      </a>
      <a href="html/seattle_classic.html">
        
      </a>
      <a href="html/seattle_shuffle.html">
        
      </a>
    </p>
  </body>

```

```
<a href="html/seattle_downtown.html">  
    
</a>  
</p>  
  
<h2>Birmingham, England</h2>  
<p>  
  Here are some iPod photos around Birmingham. We've obviously got some  
  passionate folks over here who love their iPods. Check out the classic  
  red British telephone box!  
</p>  
  
<p>  
<a href="html/britain.html">  
    
</a>  
<a href="html/applestore.html">  
    
</a>  
</p>  
</body>  
</html>
```

For each thumbnail image, wrap an `<a>` element around it.
Just be careful to get the right `href` in each link!

Add these `<a>` elements to your “index.html” file.
Save, load into your browser, and check out myPod!



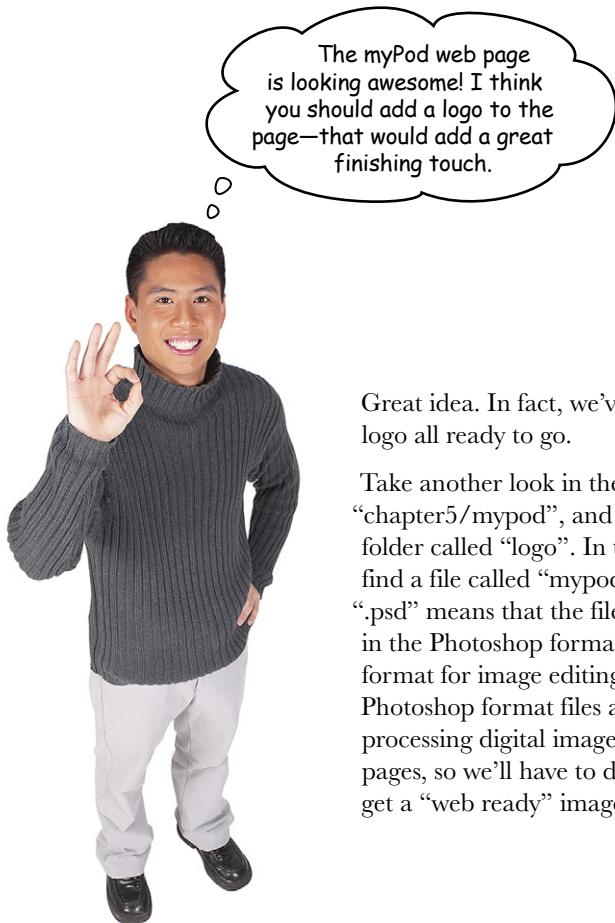
there are no
Dumb Questions

Q: When we put an `<a>` element around text, we get an underline. Why don't we get something equivalent with images?

A: Actually, Internet Explorer puts a border around an image to show it is linked. (Our browser, Safari, doesn't do that.) If your browser puts a border around or a line under your linked images, and you don't like it, hold on a few more chapters and you'll learn how to change that with CSS. Also notice that when you pass your mouse over an image, your cursor will change to indicate you can click on the linked image. In most cases your users will know an image is linked by context and by the mouse cursor, even if there's no border.

Q: Can't we just link to the JPEG image directly without all those HTML pages? I thought the browser was smart enough to display images by themselves.

A: You're right; you could link directly to the image, like this: ` ... `. If you did that and clicked on the link, the browser would display the image by itself on a blank page. In general, though, linking directly to an image is considered bad form, because you usually want to provide some context for the images you are displaying.



Great idea. In fact, we've got a myPod logo all ready to go.

Take another look in the folder “chapter5/mypod”, and you'll find a folder called “logo”. In that folder you'll find a file called “mypod.psd”. The “.psd” means that the file has been saved in the Photoshop format, a common format for image editing software. But Photoshop format files are meant for processing digital images, not for web pages, so we'll have to do some work to get a “web ready” image from it.

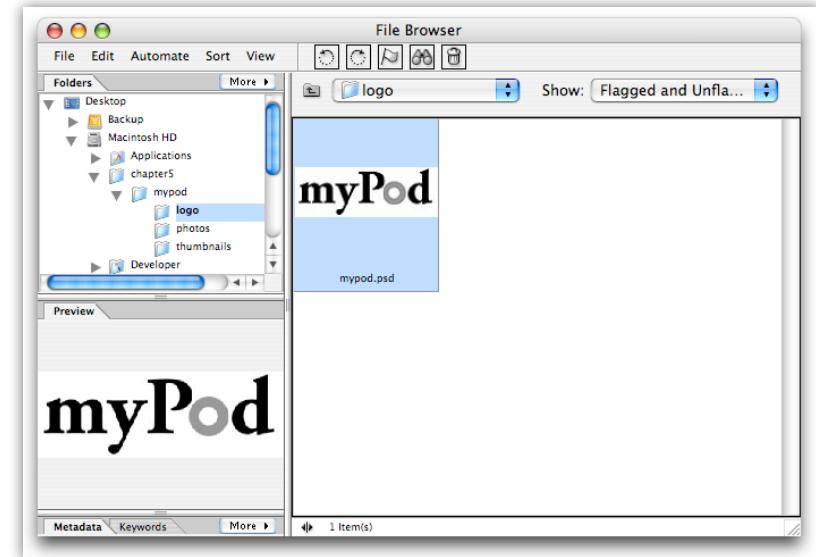
Many photo editing applications understand .psd files, so even if you don't have Photoshop Elements, follow along for the next few pages. If your application can't open the “.psd” file, you'll find the images from each step in the “logo” folder.

Open the myPod logo

Let's check out the myPod logo: open up the file "mypod.psd" in the "chapter5/mypod/logo" folder in Photoshop Elements:

You'll find the "logo" folder in the "chapter5/mypod" folder.

If your photo editing application won't open the file, follow along anyway—the same principles apply for other formats as well.



A closer look...

Nice logo; it's got some typography combined with two circles, one gray and one white (obviously inspired by the click-wheel controls on the classic iPod).

But what is that checkered pattern in the background? That's the way most photo editing applications show you areas that are transparent. Keep all that in mind as we choose a graphic format for the logo...

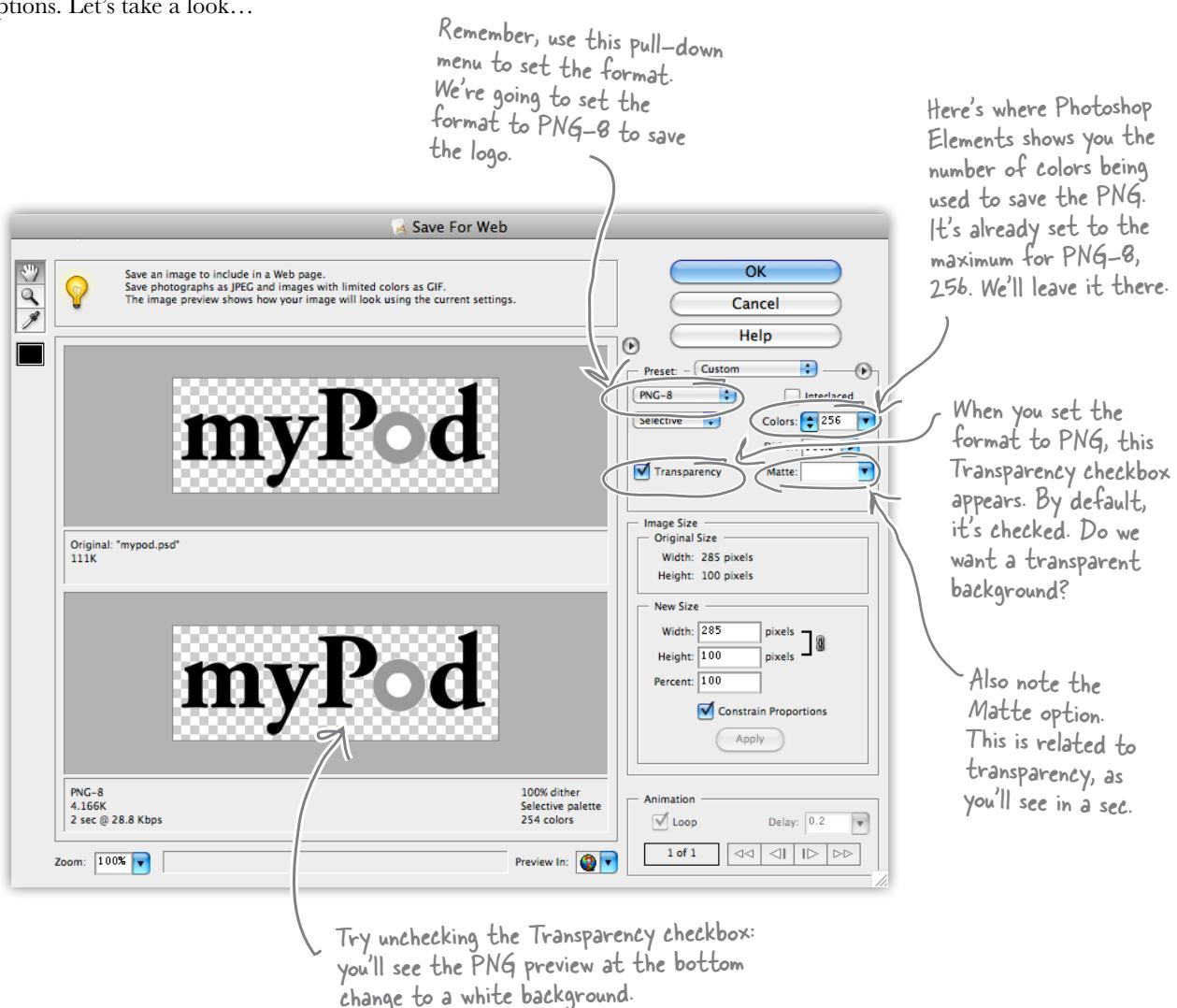


Whenever you see this checkered pattern, that indicates a transparent area in the image.

What format should we use?

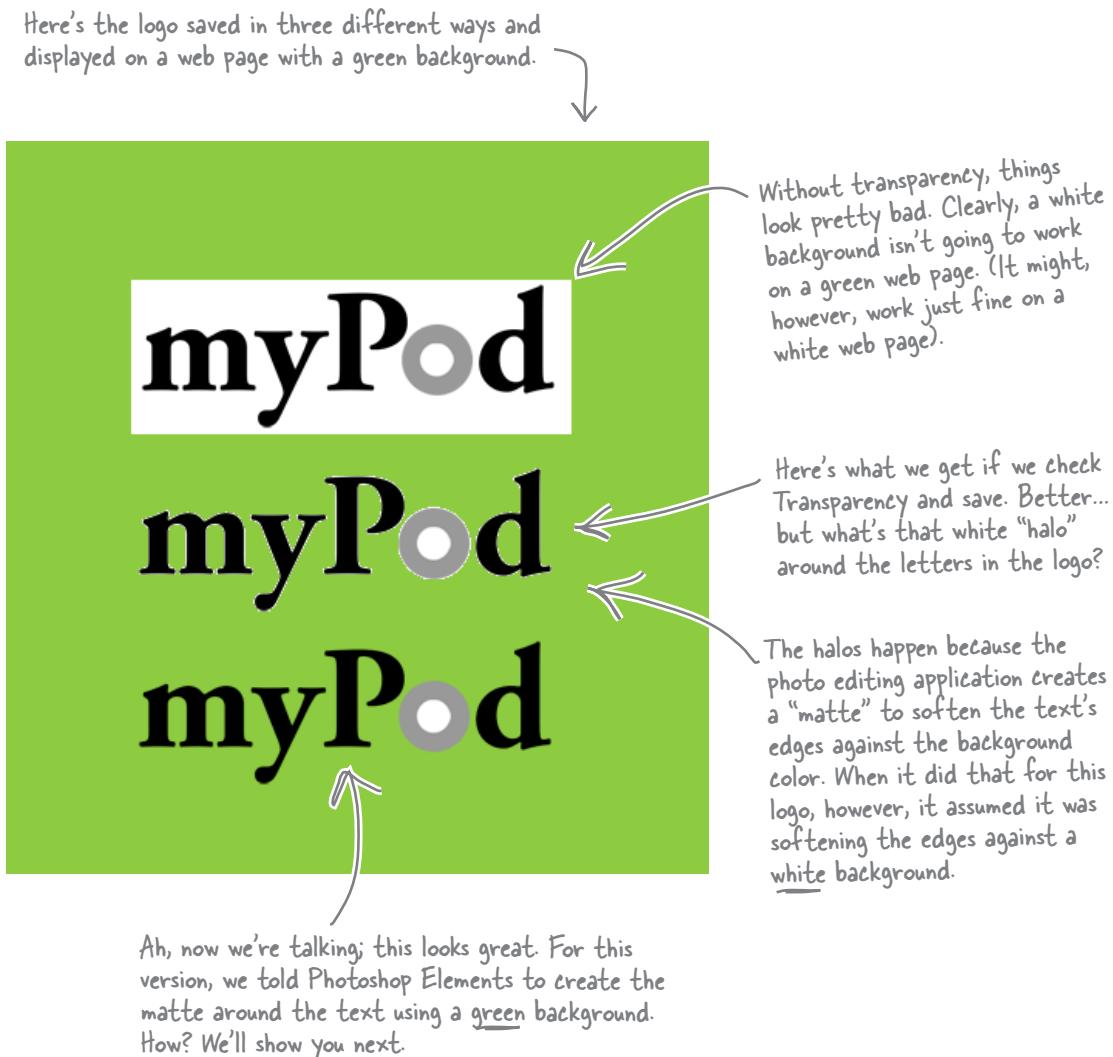
You already know that we have a couple of options in deciding how to save this image: we could use JPEG, PNG, or GIF. This logo uses only three colors, text, and some geometric shapes. From what you've learned about the two formats, you're probably leaning toward PNG or GIF. Either would be fine; the PNG might be a slightly smaller file at the same quality, so we'll go with PNG. And, because we only have three colors, we'll be safe using PNG-8 which allows only 256 colors, so using this format will reduce the file size even more.

So, go ahead and choose the “Save for Web” option under the File menu, and then choose PNG-8 in the format drop-down. You’ll see we have a few more options. Let’s take a look...



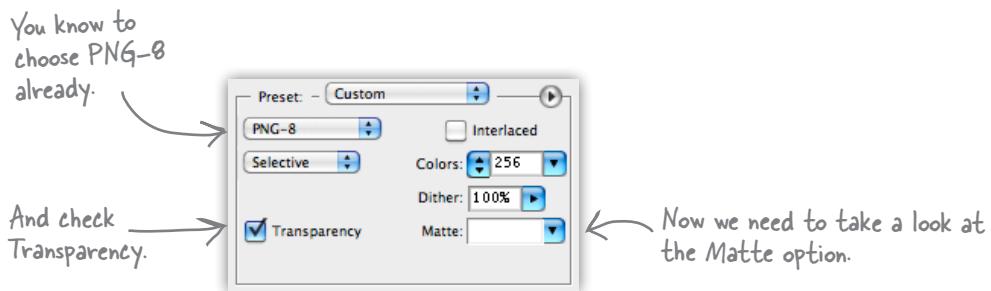
To be transparent, or not to be transparent? That is the question...

The myPod logo is going to be placed on a light green background, so you might think that transparency is going to be a good thing, right? Well, let's compare how the logo looks using a few options in the "Save for Web" dialog:



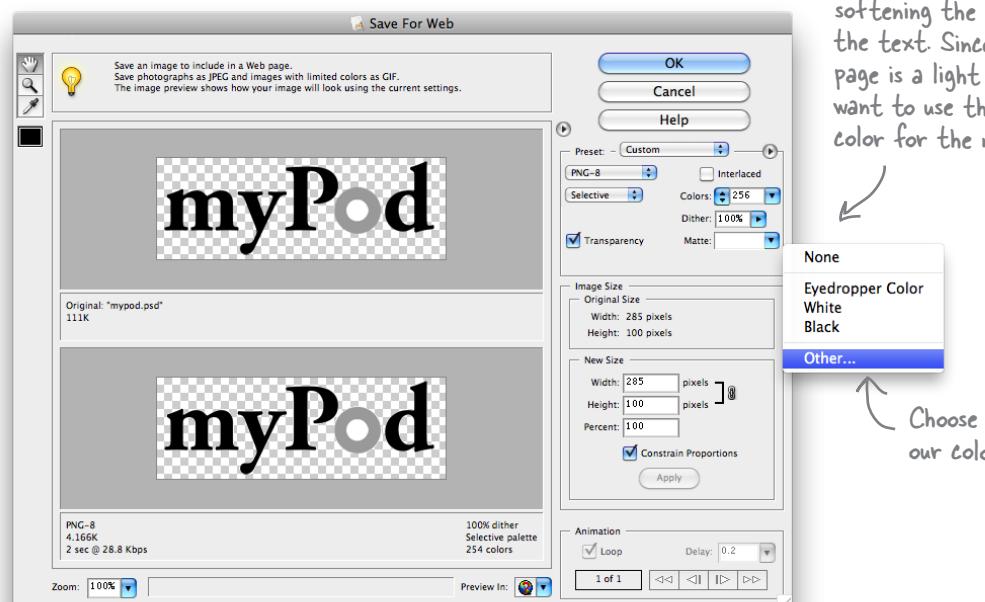
Save the transparent PNG

You know you want a transparent PNG version of the logo, and you also know we'll need to use a matte to prevent the halos around the text. Let's check out the PNG panel of the "Save for Web" dialog.



The Matte option allows you to select the color for the matte around the text. And we want that to be the color of the web page background.

The Matte option supplies the color for softening the edges of the text. Since the web page is a light green, we want to use the same color for the matte.



what is the background color?

Wait, what is the color of the web page background?

Remember that **Ready Bake CSS** in the myPod “index.html” file? That CSS is what sets the background color of the page to light green. And that’s where we can get the color:

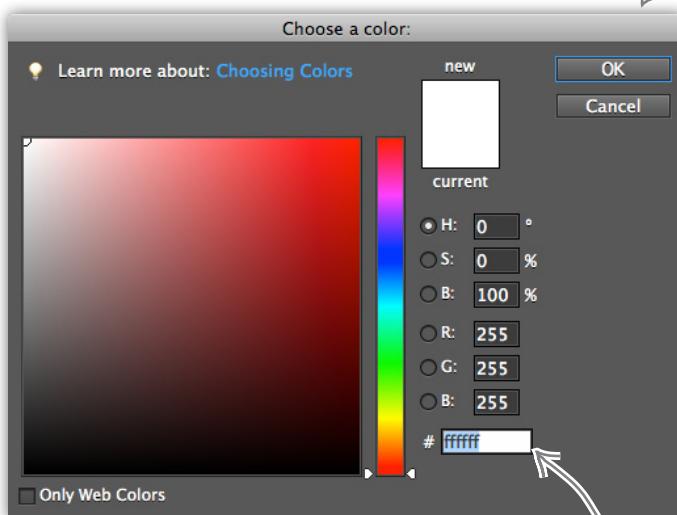
```
<style type="text/css">  
    body { background-color: #eaf3da; }  
</style>
```

Here's the background color right here.

What? You can't tell that's light green? For now, take our word for it; we'll come back to this in a few chapters and explain all about colors.

Set the matte color

When you click on the Matte pull-down menu and choose the “Other...” menu option, Photoshop Elements will bring up the Color Picker dialog.

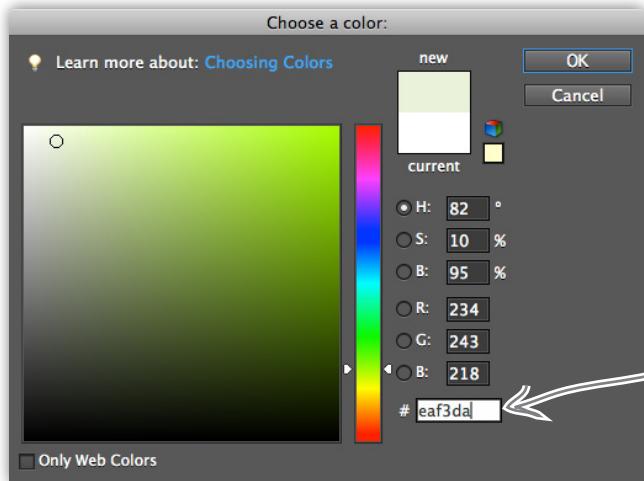


The Color Picker gives you a lot of different ways to choose the matte color. We just want to set it to the background color of the web page, and we already know that is eaf3da...

...which is going to go right here.

Set the matte color, continued

Go ahead and enter the color, “eaf3da”, into the “Color Picker” dialog box. You’ll see the color change to the background color of the myPod page.

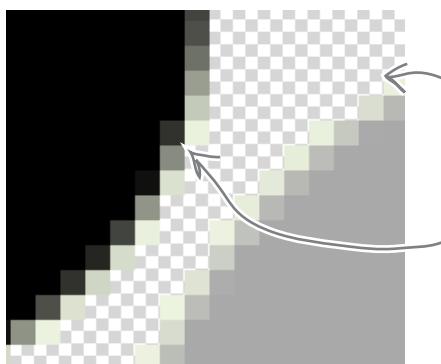


Type these letters in right here.
This box is designed specifically for
colors written in the web format.
You can type the letters in upper-
or lowercase; it doesn't matter.

Once you've typed the color
into the Color Picker, click
OK and it will apply the
change to the logo.

Check out the logo with a matte

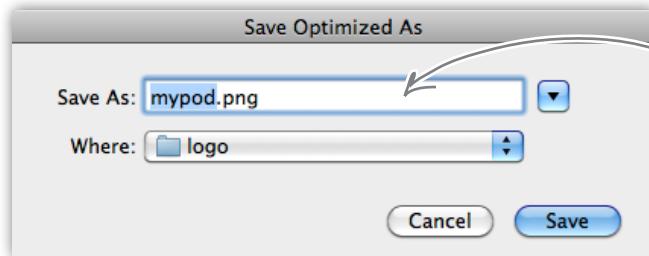
Now take a close look at the logo again in the preview pane. You'll see Photoshop Elements has added a light green matte around the hard edges, which will give the myPod logo text a softer, more polished look when the logo is in the web page.



Now, when you look close up at the
logo, you'll see the matte matches the
green color in the background of the
myPod web page.

Save the logo

Okay you've made all the adjustments you need to in the "Save for Web" dialog, so go ahead and click OK to save the image as "mypod.png".



Elements will automatically change the extension of your filename to ".png". Save the image as "mypod.png" in the "logo" folder.

Add the logo to the myPod web page

Now all you need to do is add the logo to the myPod web page. We'll add it to the top so it appears above the website description and iPod images. That way, it's the first thing your visitors see when they come to your myPod page.

```
<html>
  <head>
    <title>myPod</title>
    <style type="text/css">
      body { background-color: #eaf3da; }
    </style>
  </head>
  <body>
    <p>
      
    </p>
    <h1>Welcome to myPod</h1>
    .
    .
  </body>
</html>
```

Add the logo image at the top of the myPod web page. Remember to use the correct relative path for the logo, in the "logo" folder, and add an alt attribute that describes the image.

Rest of "index.html" HTML here...

And now for the final test drive



Let's test this puppy! Reload the web page in the browser and see how your myPod transparent PNG logo works.

myPod

Welcome to myPod

Welcome to the place to show off your iPod, wherever you might be. Wanna join the fun? All you need is any iPod, from the early classic iPod to the latest iPod Nano, the smallest iPod Shuffle to the largest iPod Video, and a digital camera. Just take a snapshot of your iPod in your favorite location and we'll be glad to post it on myPod. So, what are you waiting for?

Seattle, Washington

Me and my iPod in Seattle! You can see the Space Needle. You can't see the 628 coffee shops.

Birmingham, England

Here are some iPod photos around Birmingham. We've obviously got some passionate folks over here who love their iPods. Check out the classic red British telephone box!

And it works—all that hard work paid off. You have a great-looking logo on your myPod web page.

Excellent work.
The logo looks great.
You've got a kick-ass
myPod website here!



^{there are no} Dumb Questions

Q: Do I really need to know all this stuff about image formats to write good web pages?

A: No. You can build great web pages without any images. However, images are a big part of the Web, so some knowledge of how images work can really help. Sometimes just an image or two makes the difference between a good page and a great one. There's a lot to know about images, but it's easy to learn as you go.

Q: Why does the text need its edges softened?

A: Check out the two versions of the myPod logo below:



You'll see the top version has very hard, jagged edges and is less readable. This is the way text is displayed by default on a computer screen. The second version has had its edges softened using a technique called *anti-aliasing*. Words that are anti-aliased on a computer screen are more readable and more pleasant to the eye.

Q: So where does the matte come in?

A: The process of anti-aliasing softens the edges relative to the background color. If you put the bottom version of the logo (from the previous Q&A) against a colored background, you'd see it has white edges in it. The Matte option in Photoshop Elements allows you to specify the color of the background that the text will be placed on, so when the text is softened it is done so against that color.

Q: Does this technique just work for text?

A: No, it works for any lines in your graphics that might result in "jaggies." Check out the circle in the myPod logo; it was matted too.

Q: Why can't I just make the logo background color solid and match the color to the web page?

A: You could do that too, but there is one disadvantage: if there are other things in your web page that are showing through the transparency, then they won't be seen with the solid color version. You haven't seen any examples of this yet, but when we get into CSS, you will.

Q: What if I change my background color after I make the matted version?

A: A slight variation in your background color probably wouldn't be noticeable; however, if you change the color dramatically, you'll have to recreate the PNG with a new matte color.

If you're placing a transparent image in your web page, make sure the matte color of the image matches the background color of your web page.

You can use **PNG** or **GIF** format for your transparent image.



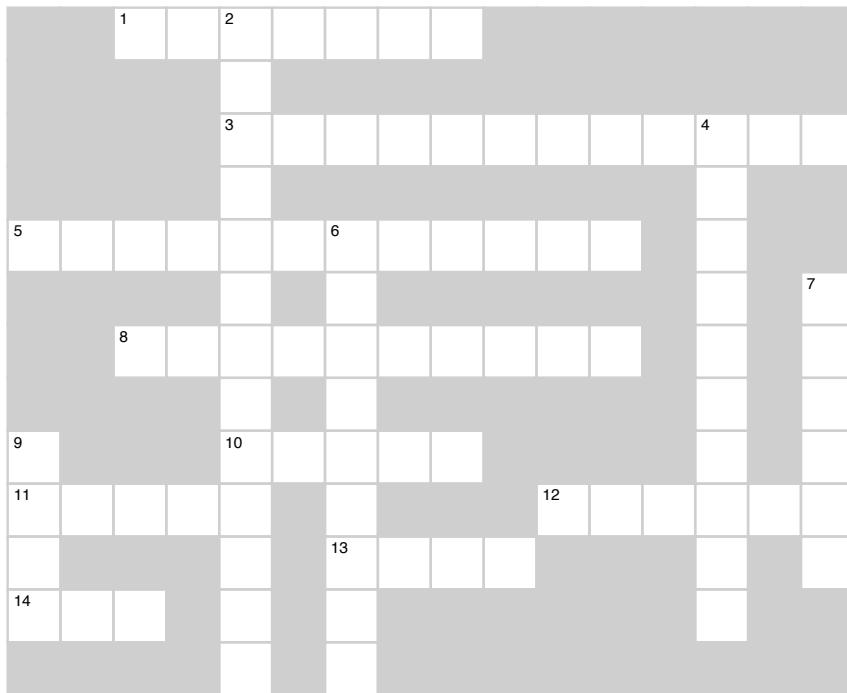
BULLET POINTS

- Use the `` element to place images in your web page.
- Browsers treat `` elements a little differently than other HTML elements; after reading the HTML page, the browser retrieves each image from the web server and displays it.
- If you have more than a couple of large images on a web page, you can make your web page more usable and faster to download by creating thumbnails—small images that the user can click on to see the large version of the image.
- The `` element is an inline element, which means that the browser doesn't put a linebreak before or after an image.
- The `src` attribute is how you specify the location of the image file. You can include images from your own site using a relative path in the `src` attribute, or images from other sites using a URL.
- The `alt` attribute of an `` element is a meaningful description of the image. It is displayed in some browsers if the image can't be located, and is used by screen readers to describe the image for the visually impaired.
- A width of less than 800 pixels is a good rule of thumb for the size of photo images in a web page. Most photo images that are created by digital cameras are too large for web pages, so you'll need to resize them.
- Photoshop Elements is one of many photo editing applications you can use to resize your images. You can also use one of many free online tools to resize images. Just search for "free online image editor."
- Images that are too large for the browser make web pages difficult to use and slow to download and display.
- JPEG, PNG, and GIF are the three formats for images that are widely supported by web browsers.
- The JPEG format is best for photographs and other complex images.
- The GIF or PNG format is best for logos and other simple graphics with solid colors, lines, or text.
- JPEG images can be compressed at a variety of different qualities, so you can choose the best balance of quality and file size for your needs.
- The GIF and PNG image formats allow you to make an image with a transparent background. If you put an image with a transparent background in a web page, what's behind the image, such as the background color of the page, will show through the transparent parts of the image.
- GIF and PNG are lossless formats, which means the file sizes are likely to be larger than JPEG.
- PNG has better transparency control than GIF, and allows many more colors than GIF, which is limited to 256.
- PNG has three different size options: PNG-24 (supports millions of colors), PNG-16 (supports thousands of colors), and PNG-8 (supports 256 colors), depending on your needs.
- In Photoshop Elements, use the Matte color menu in the "Save for Web" dialog to choose the right color for softening the edges of your transparent PNG or GIF image.
- Images can be used as links to other web pages. To create a link from an image, use the `` element as the content of an `<a>` element, and put the link in the `href` attribute of the `<a>` element.



HTMLcross

It's time to give your right brain another break and put that left brain to work. All these words are HTML-related and from this chapter.



Across

1. With JPEG, you can control this.
3. Most web browsers retrieve images this way.
5. PNG and GIF have it, JPEG doesn't.
8. Miles you can draw with a pencil.
10. Web server makes a request for each one of these.
11. Smallest element on a screen.
12. You used Photoshop Elements to do this to images.
13. Lovable MP3 player.
14. Better for solid colors, lines, and small text.

Down

2. The alt attribute improves this.
4. Small images on a page.
6. Technique for softening edges of text.
7. The larger the image, the _____ it takes to transfer it.
9. Better for photos with continuous tones.

WHICH IMAGE FORMAT? SOLUTION

Congratulations: you've been elected "Grand Image Format Chooser" of the day. For each image below, choose the format that would best represent it for the Web.

JPEG or PNG or GIF



A photo with lots of continuous shades of gray.



Only a couple of colors with some text; definitely a PNG or GIF. No transparency? PNG might yield a smaller file.



A photo with lots of colors; definitely a JPEG or PNG; and if you want a transparent background, go with PNG.



Just a simple black and white icon; a PNG or GIF. If you need transparency, you might want anti-aliasing on the edges, and PNG would be better for that.



This image is borderline. It has lots of continuous colors (JPEG), but is also slightly geometric (GIF) and you may want to use this in a way that requires transparency (PNG).



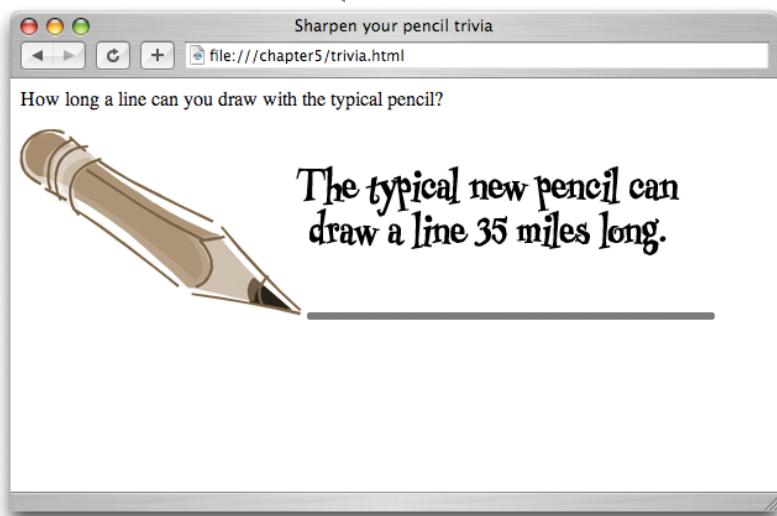
Sharpen your pencil Solution

Here's a "Sharpen your pencil" that is actually about pencils (oh, and images too). This exercise involves a bit of trivia: *Given a typical, brand-new pencil, if you drew one continuous line with it, using the entire pencil up, how long would the line be?*

What's that got to do with images? To find the answer, you had to write some HTML. The answer to this trivia is contained in the image that is at the URL: <http://wickedlysmart.com/hfhtmlcss/trivia/pencil.png>. Your job was to add an image to this HTML and retrieve the answer. Here's our solution.

```
<html>
  <head>
    <title>Sharpen your pencil trivia</title>
  </head>
  <body>
    <p>How long a line can you draw with the typical pencil?</p>
    <p>
      
    </p>
  </body>
</html>
```

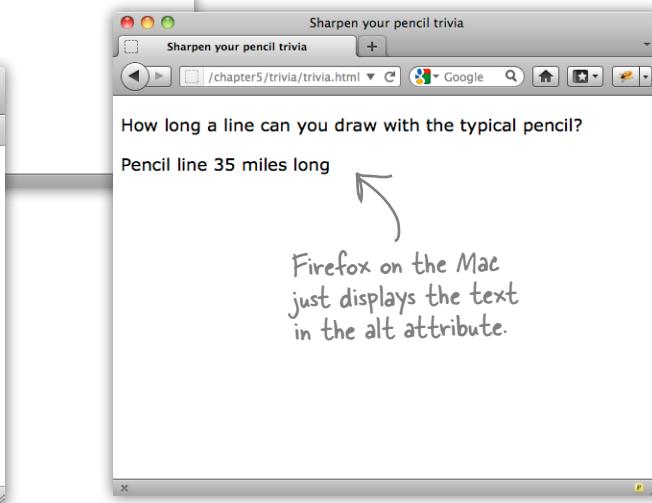
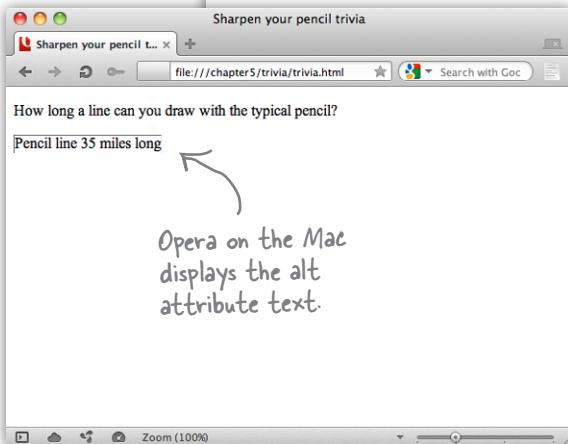
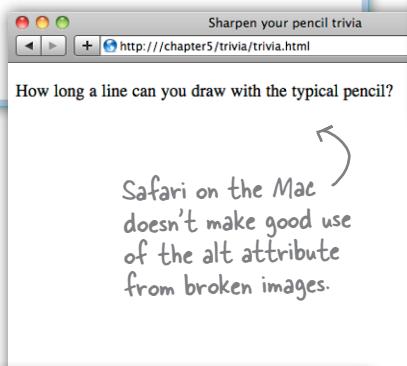
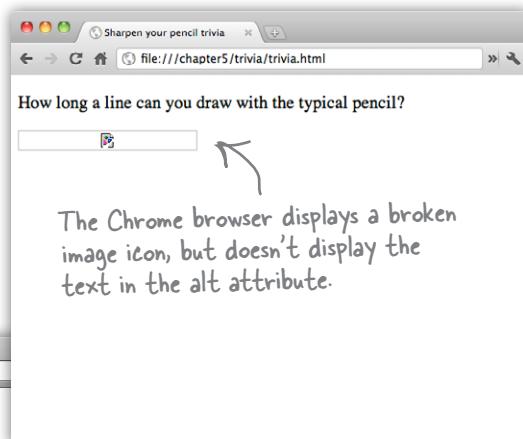
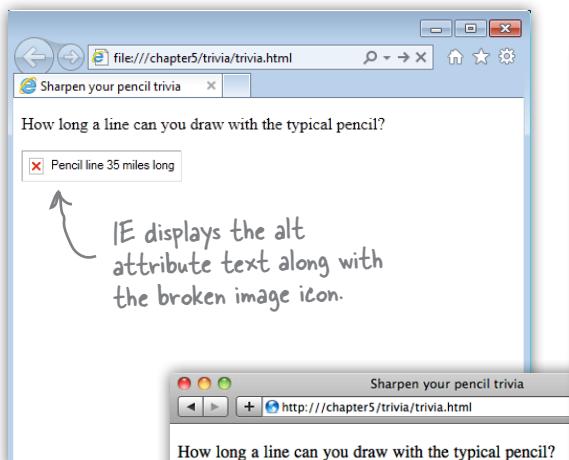
If you put the image here, you can see
the answer when you load the page.



Source: <http://www.papermate.com>



Here are the results of having a broken image in a few different browsers. In most cases, the browser is able to use the extra alt attribute information to improve what is displayed. Why do we care? After all, this is an error in a web page; we should just fix it, right? Well, in the real world, things are often not ideal; sometimes things break, Internet connections go bad in the middle of a page load, or visually impaired users need to hear what is in the image, because they can't see it.

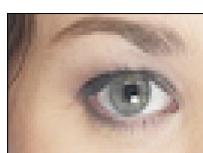


* * WHICH IMAGE FORMAT? *

SOLUTION

Your task this time: open the file “chapter5/testimage/eye.jpg” in Photoshop Elements. Open the “Save for Web” dialog and fill in the blanks below by choosing each quality setting for JPEG (Low, Medium, High, etc.), and also try PNG-24 and GIF. You’ll find this information in the preview pane below the image. Once you’ve finished, determine which setting makes the most sense for this image.

Note that your numbers may differ depending on the version of software you are using.

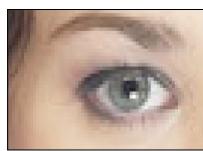


Format	Quality	Size	Time	Winner
--------	---------	------	------	--------

<u>PNG-24</u>	<u>N/A</u>	<u>32K</u>	<u>13 seconds</u>	<input type="checkbox"/>
---------------	------------	------------	-------------------	--------------------------



<u>JPEG</u>	<u>Maximum</u>	<u>21K</u>	<u>8 seconds</u>	<input type="checkbox"/>
-------------	----------------	------------	------------------	--------------------------



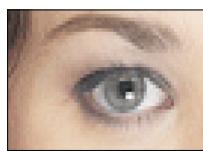
<u>JPEG</u>	<u>High</u>	<u>6K</u>	<u>3 seconds</u>	<input type="checkbox"/>
-------------	-------------	-----------	------------------	--------------------------



<u>JPEG</u>	<u>Medium</u>	<u>3K</u>	<u>2 seconds</u>	<input checked="" type="checkbox"/>
-------------	---------------	-----------	------------------	-------------------------------------



<u>JPEG</u>	<u>Low</u>	<u>2K</u>	<u>1 second</u>	<input type="checkbox"/>
-------------	------------	-----------	-----------------	--------------------------



<u>GIF</u>	<u>N/A</u>	<u>22K</u>	<u>9 seconds</u>	<input type="checkbox"/>
------------	------------	------------	------------------	--------------------------

Did you notice how the image quality degrades as you go from JPEG Maximum to Low?



Is the winner really Medium? Not necessarily. It all depends on what your needs are. If you want a really high-quality image, then you might want Very High. If you want the fastest possible site, then try Low. We've chosen Medium because it is a nice tradeoff in size versus the quality of the image. You may think Low is good enough, or that it's worth bumping the quality up to High. So, it's all very subjective. One thing is for sure, however: PNG and GIF don't work very well for this image (which should not be a surprise).



Exercise Solution

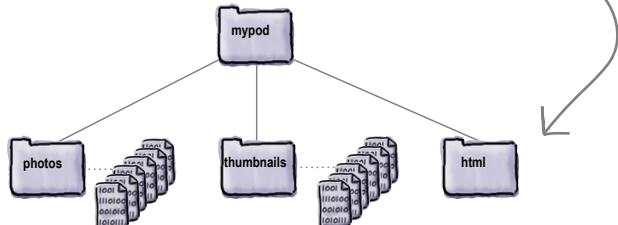
If you look in the “html” folder with the chapter examples, you’ll find all of the single photo pages already there, except one: the page for “seattle_downtown.jpg”. Create a page called “seattle_downtown.html” in the “html” folder, and test it out. Get this working before you move on.

Here’s the answer:

Here’s the HTML; this file should be called “seattle_downtown.html”.

```
<html>
  <head>
    <title>myPod: Seattle Downtown</title>
    <style type="text/css"> body { background-color: #eaf3da; } </style>
  </head>
  <body>
    <h1>Downtown Seattle</h1>
    <p>
      
    </p>
  </body>
</html>
```

This file should go in the “html” folder under “mypod”.



Here’s the test drive.





HTMLcross Solution



Sharpen your pencil Solution

Here's how you add the image "seattle.jpg" to the file "index.html".

<h2>Seattle, Washington</h2>

<p>

Me and my iPod in Seattle! You can see rain clouds and the Space Needle. You can't see the 628 coffee shops.

</p>

<p>

```
  
</p>
```

6 standards and all that jazz

Getting Serious with HTML



What else is there to know about HTML? You're well on your way to mastering HTML. In fact, isn't it about time we move on to CSS and learn how to make all this bland markup look fabulous? Before we do, we need to make sure your HTML is really ready for the big leagues. Don't get us wrong, you've been writing first-class HTML all along, but there are just a few extra things you need to do to make it "industry standard" HTML. It's also time you think about making sure you're using the latest and greatest HTML standard, otherwise known as HTML5. By doing so, you'll ensure they'll display more uniformly across all browsers (at least the ones you'd care about), not to mention, they'll play well with the latest i-Devices (pick your favorite). You'll also have pages that load faster, pages that are guaranteed to play well with CSS, and pages that are ready to move into the future as the standards grow. Get ready, this is the chapter where you move from web tinkerer to web professional.



Jim: Ready for prime time?

Frank: Yeah, you know, make sure it's totally legit and ready for HTML5.

Jim: Our HTML is just fine...here, look at it in the browser. It looks beautiful.

Joe: Yeah, that's what I think...he's just trying to give us another thing to do.

Frank: Actually guys, I hate to admit it, but I think the boss is right on this one.

Jim, Joe: Huh?

Frank: Up until now we've pretty much ignored the fact that there are standards for this stuff. Not to mention there are different versions of HTML, like HTML 4.01, and now, HTML5. Are we doing everything we need to make sure we've got HTML5 covered?

Joe: Come on, this is just going to mean even more work. We've already got enough to do. Really, the page looks fine; I've even tested it on some of the newer devices.

Frank: That may be, but what I'm saying is that I think this will help us do less work in the future.

Joe: Oh yeah? How so?

Frank: Well, if we make sure our HTML is up-to-date with current standards, we won't have to make as many changes down the road. We should also make sure everything else is correct; you know, our syntax and all that. There are so many different browsers and versions of those browsers that if we're making mistakes in our HTML, then all bets are off in terms of how our pages will look in different browsers. And when we start adding presentation to HTML with CSS, the differences will get even more dramatic if our HTML isn't up to snuff.

Joe: So, by making sure we're adhering to the "standard," we'll have a lot fewer problems with our pages displaying incorrectly for our customers?

Frank: Right.

Jim: If it reduces the number of 3 a.m. calls I get, then that sounds like a good idea to me.

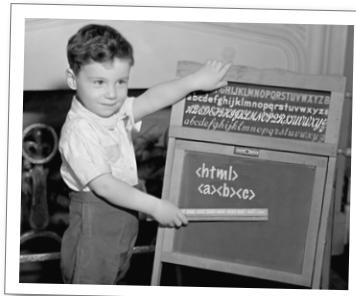
Joe: All right, how do we start? Don't we adhere to the standards now? What's wrong with our HTML?

Frank: Maybe nothing, but the boss wants to be current with HTML5, so we need to figure out which version of HTML we're using and if it's not HTML5, what we need to do to get there. And, when we're done, life should be much easier when we start using CSS.



Browsers all do a pretty good job of consistently displaying your pages when you write correct HTML, but when you make mistakes or do nonstandard things in your HTML, pages are often displayed differently from one browser to another. Why do you think that is the case?

A Brief History of HTML



HTML 1.0-2.0

These were the early days; you could fit everything there was to know about HTML into the back of your car. Pages weren't pretty, but at least they were hypertext enabled. No one cared much about presentation, and just about everyone on the Web had their very own "home page." Even a count of the number of pencils, paperclips, and Post-it notes on your desk was considered "web content" back then (you think we're kidding).

HTML 3

The long, cold days of the "Browser Wars." Netscape and Microsoft were duking it out for control of the world. After all, he who controls the browser controls the universe, right?

At the center of the fallout was the web developer. During the wars, an arms race emerged as each browser company kept adding their own proprietary extensions in order to stay ahead. Who could keep up? And not only that, back in those days, you had to often write two separate web pages: one for the Netscape browser and one for Internet Explorer. Not good.

HTML 4

Ahhh...the end of the Browser Wars and, to our rescue, the World Wide Web Consortium (nickname: W3C). Their plan: to bring order to the universe by creating the ONE HTML "standard" to rule them all.

The key to their plan? Separate HTML's structure and presentation into two languages—a language for structure (HTML) and a language for presentation (CSS)—and convince the browser makers it was in their best interest to adopt these standards.

But did their plan work? Uh, almost...with a few changes (see HTML 4.01).

1989

1991

1995

1998

Starting with this chapter, our goal is to write proper HTML5. As always, the world keeps moving, so we'll also talk about where things are going.



HTML 4.01

The good life: HTML 4.01 entered the scene in 1999, and was the “must have” version of HTML for the next decade.

4.01 wasn't really a big change from 4.0; just a few fixes were needed here and there. But compared to the early days of HTML (when we all had to walk barefoot in six feet of snow, uphill both ways), HTML 4.01 allowed us all to sleep well at night knowing that almost all browsers (at least the ones anyone would care about) were going to display our content just fine.



XHTML 1.0

Just as we were all getting comfortable, a shiny object distracted everyone. That shiny object was XML. In fact, it really distracted HTML, and the two got hitched in a shotgun marriage that resulted in XHTML 1.0.

XHTML promised to end all the woes of the Web with its adherence to strictness and new way of doing things.

The only problem was, ^{most} people ended up hating XHTML. They didn't want a new way to write web pages, they just wanted to improve what they already had with HTML 4.01. Web developers were far more interested in HTML's flexibility than XHTML's strictness. And, more and more, these developers wanted to spend their time creating web pages that felt more like applications than documents (more on web apps later)...



HTML5

Of course, with no support from the community, the marriage didn't end well and was replaced by new version of HTML named HTML5. With its support for most of the HTML 4.01 standard, and new features that reflect the way the Web has grown, HTML5 is what developers were looking for. And, with features like support for blog-like elements, new video and graphic capabilities, and a whole new set of capabilities aimed at building web applications, HTML5 was destined to become the standard.

To be honest, the divorce of HTML and XML took a lot of people by surprise, leading to confusion about what HTML5 actually *is* for a while. But that's all been sorted out, so read on to find out what HTML5 means to you, and how you can join in the fun.

1999

2001

2009

2012

????

And what will happen in the future? Will we all be going to work in flying cars and swallowing nutrition pills for dinner? Keep reading to find out.



The Browser Exposed

This week's interview:

**Why do you care so much about
which HTML version I'm using?**

Head First: We're glad to have you here, Browser. As you know, "HTML versions" have become a popular issue. What's the deal with that? You're a web browser, after all. I give you HTML and you display it the best you can.

Browser: Being a browser is tough...there are a lot of web pages out there, and many are written with old versions of HTML or with mistakes in their markup. Like you said, my job is to try to display every single one of those pages, no matter what.

Head First: So what's the big deal? You seem to be doing a pretty good job of it.

Browser: In some cases, sure, but have you ever looked at your pages on a lot of different browsers? When you are using old or incorrect markup, your page may look great on one browser, but not so great on another.

Head First: Really? Why is that? Don't you all do the same thing?

Browser: We do a great job of doing the same thing, *when we're displaying correct and up-to-date pages*. Like I said, when you venture into pages that aren't written well, then things get a lot dicier. Here's why: all of us browsers have the HTML specification to tell us how to display correct HTML, but when it comes to incorrect HTML, we just wing it. So, you might get very different behaviors on different browsers.

Head First: Ahh. So, what's the solution to this mess? We definitely want our pages to look good.

Browser: Easy. Tell me up front which version of HTML you're using. You'd be surprised how many pages don't even do that. And make sure your page doesn't contain any errors; you know, mismatched markup tags, that kind of thing.

Head First: How do we tell you which version we're using? Especially now that we're all moving on to HTML5.

Browser: Well, HTML5 is actually making things a little simpler.

Head First: Really? How is a new version of HTML helping? I would have thought yet another version would just make things even more difficult.

Browser: It's true that any new version of a language causes growing pains as everyone tries to catch up with the latest standard. But HTML5 simplifies the way you tell me the kind of HTML you're using. The HTML5 standard is also documenting many of the errors that can occur in web pages, so that all the browsers can be more consistent about how they handle those errors.

Head First: Oh, so does that mean we don't have to worry about making errors when we're writing our HTML?

Browser: No! Just because we can handle errors better doesn't mean you can be sloppy. You still want your page to be consistent with the standard and written without errors. If you don't, you might get inconsistent results across browsers, and let's not forget the browsers on mobile devices, too.

Head First: Back to how we tell you what version we're using?

Browser: Yeah, that used to be a total pain in the...

Head First: Uh, watch it, this is a PG-rated audience, and we're running out of time, quickly!

Browser: Okay, you can tell me all about the version of HTML you're using with a **doctype**. It's a little bit of "markup" you can use that goes at the very top of your HTML file. So, given we're out of time, go check it out!



HTML Archaeology



We did some digging and found some old HTML 4.01 and XHTML 1.1 pages. These pages use a doctype, at the very top of the HTML file, to tell the browser which version of HTML they're using. We've snipped out a couple of doctypes for you to look at. Check them out below...

This is specifying a document type for this page to the browser.

This means that `<html>` is the root (first) element in your page.

This just means the HTML 4.01 standard is publicly available.

This part says we're using HTML version 4.01 and that HTML markup is written in English.

You can type this all on one line, or if you want, you can add a return where we did. Just make sure you only press Return in between the parts in the quotes.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"  
      "http://www.w3.org/TR/html4/strict.dtd">
```

Notice that this is NOT an HTML element. It has a “!” after the “<” at the beginning, which tells you this is something different.

This points to a file that identifies this particular standard.

Just like the HTML DOCTYPE, this is a public document type.

It's still a version of HTML—an XML version.

It's for the XHTML 1.1 version of XHTML.

```
<!DOCTYPE html  
      PUBLIC "-//W3C//DTD XHTML 1.1//EN"  
      "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

For more on XHTML, check out the appendix.

And it has a URL pointing to the definition of XHTML 1.1.



Sharpen your pencil

Rather than tell you the doctype definition for HTML5, we thought you might want to have fun working it out on your own. Take another look at the HTML 4.01 doctype definition below:

Remember, this is the doctype for "html".

And this means this standard is publicly available.

This part says we're using HTML version 4.01 and that this markup is written in ENglish.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">
```

This points to a file that identifies this standard.

Remember, the doctype definition belongs at the top of your HTML file and tells the browser the type of your document—in this case, HTML 4.01. By using a doctype, the browser is able to be more precise in the way it interprets and renders your pages.

So, using your deductive powers, what do you think the doctype definition for HTML5 looks like? Write it here (you can refer back to your answer when we cover this on the next page, and no peeking at the answer!):

.....
.....
.....
.....
.....

Your answer goes here. ↗

The new, and improved, HTML5 doctype

Okay, get ready for it. Here's the HTML5 doctype:

```
<!doctype html>
    ↑
And it's really simple!
```

It's just one line; don't miss it.

How close was your answer in the Sharpen Your Pencil? This is much simpler, wouldn't you say? And, wow, you might even be able to remember it without having to look it up everytime you need a doctype.

Our sympathies
to those who had
the old doctype
tattooed on
their palms to
remember it.

Wait, isn't this supposed to tell
the browser the version? Where's the
version number? Is that a typo?



Good point. No, it's not a typo, and let's step through why: you know the doctype used to be a complicated mess full of version numbers and ugly syntax. But with the arrival of HTML5, the doctype was simplified so that now all we have to do is tell the browser we're using "html"; no more worrying about specific version numbers or languages or pointing to a standard.

How can that be? How can we just specify "html" without the rest? Doesn't the browser *need* that other information? Well, as it turns out, when the browser sees:

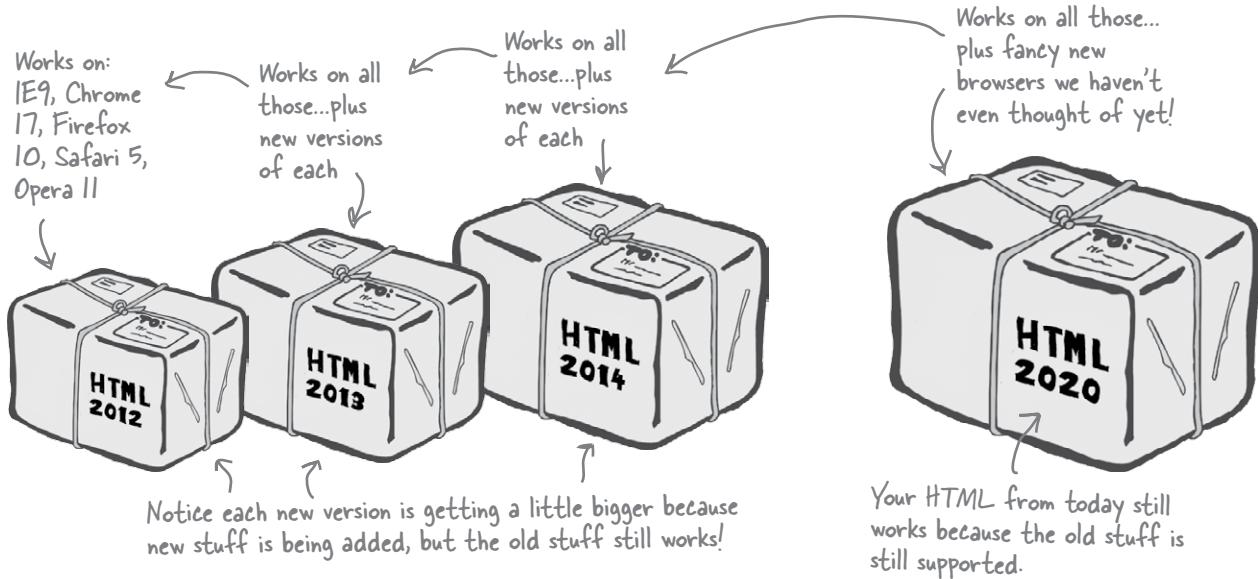
```
<!doctype html>
```

it assumes you're using standard HTML. No more getting all hung up on version numbers or where the standard is located; in fact, the HTML standard has become a "living standard," meaning that it will continue to grow and morph as needed, but without fixed version numbers. Now, you're probably thinking, "What exactly does a *living standard* mean? How is that going to work?" You'll see, on the next page...

HTML, the new “living standard”

You heard us right...rather than continue to crank out version 6, 7, 8 of HTML, the standards guys (and gals) have turned the specification into a living standard that will document the technology as it evolves. So, no more version numbers. And you can stop calling it HTML5 because it's just “HTML” from here on out.

Now, you're probably wondering how this is going to work in practice. After all, if the spec is continually changing, what does that mean for the poor browsers? Not to mention, for you, the web developer? One key to this working is *backwards compatibility*. Backwards compatibility means that we can keep adding new stuff to HTML, and the browsers will (eventually) support this new stuff, but they'll also keep supporting the old stuff. So the HTML pages you're writing today will keep working, even after new features have been added later.



there are no Dumb Questions

Q: So what happens if the spec changes tomorrow? What do I do?

A: If you're writing solid HTML today and the spec changes tomorrow to incorporate a new element, then you can just keep on doing what you're doing. It's up to you whether you want to use that new element or not.

If the spec changes something you're already doing, like the way an element or attribute works, then browsers are supposed to continue to support the old way you're using it as well as the new way. That's what “backwards compatibility” means. Now, it is obviously a good thing if existing features are changed as little as possible, and if you, as a web developer, keep up-to-date on the spec and change your pages as the spec changes, but the idea is that your HTML will continue to work as the spec changes.

Q: What exactly is a spec, anyway?

A: The specification is the document that specifies what the HTML standard is; that is, what elements and attributes are in HTML, and more. This document is maintained by the World Wide Web Consortium (W3C, for short), but anyone can contribute to it and have a say in how the standard is developed.



Adding the document type definition

Enough talk, let's get that doctype in the HTML.

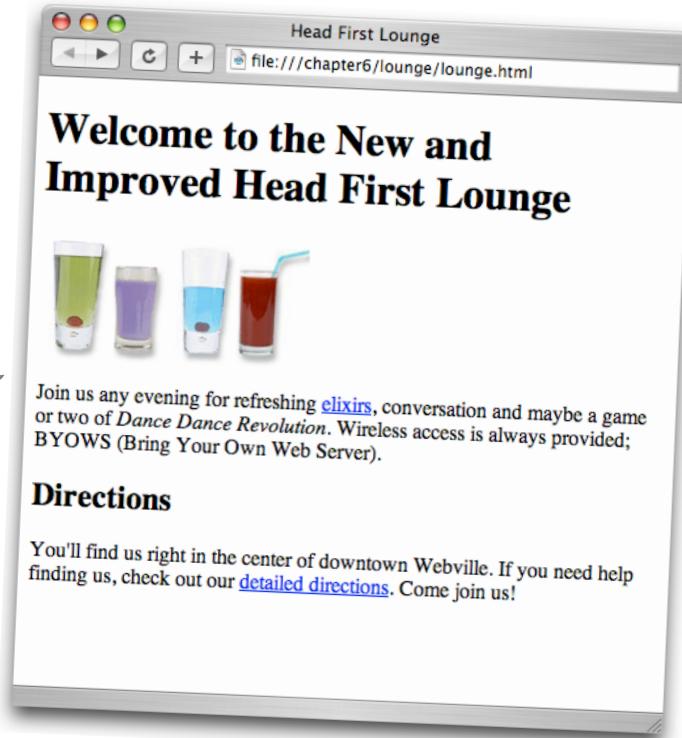
Here's the doctype line. Just add it as the very first thing in the "lounge.html" file.

You can write DOCTYPE or doctype. Both work.

```
<!doctype html>
<html>
  <head>
    <title>Head First Lounge</title>
  </head>
  <body>
    <h1>Welcome to the New and Improved Head First Lounge</h1>
    
    <p>
      Join us any evening for refreshing
      <a href="elixir.html">elixirs</a>, conversation and
      maybe a game or two of <em>Dance Dance Revolution</em>.
      Wireless access is always provided; BYOWS (Bring
      your own web server).
    </p>
    <h2>Directions</h2>
    <p>
      You'll find us right in the center of downtown
      Webville. If you need help finding us, check out our
      <a href="directions.html">detailed directions</a>.
      Come join us!
    </p>
  </body>
</html>
```

The doctype test drive

Make the changes to your “lounge.html” file in the “chapter6/lounge” folder and then load the page in your browser.



Exercise

Add a **doctype** to the “directions.html” and “elixir.html” file as well. Go ahead and give them a good test. Just like “lounge.html”, you won’t see any fireworks (but you might sleep a bit better tonight).



HTML5 Exposed

This week's interview:
What's the big deal about HTML5?

Head First: HTML5, you're the “latest and greatest” version of HTML that everyone's yammering on about, but our readers want to know what's so great about you.

HTML5: First off, I've got a bunch of new elements and some new attributes too.

Head First: We don't seem to be using any of those yet, are we?

HTML5: All the elements you're using are part of my standard now, so you're using HTML5 elements. But no, you're not using any of the *new* ones yet...

Head First: Why not? Shouldn't we be using all the newest elements as soon as possible?

HTML5: Not necessarily. Remember (from Chapter 3): always use the right element for the job! And my newest elements have specific jobs. Some of them are for adding more structure and meaning to your page. Like my new <article> element, which is specifically for things like blog posts and news articles.

Head First: We could have used that in Chapter 3 for Tony's blog, right?

HTML5: That's true...I'm sure you'll add it later on.

Head First: I'm sure our readers are wondering, since they're learning HTML in this book, if they need to go learn HTML5 instead?

HTML5: No! HTML5 is just the next evolutionary step, everything they've learned is exactly the same in HTML5. HTML5 just adds some new stuff. In fact, we should stop saying “HTML5.” I'm just the latest version of HTML, so call me HTML. Saying HTML5 at this point is just confusing.

Head First: Wait, after all the hype around HTML5, are you really suggesting we do away with your name?

HTML5: I am. You already know I'm a living standard and version numbers are dead. Well, I'm a living standard for HTML, not HTML5.

Head First: Got it. Our readers really should just continue learning HTML5—uhh sorry, HTML—and everything they've learned so far has been relevant. Not to mention all the new stuff ahead that they'll be learning is the latest and greatest HTML technology.

HTML5: Exactly.

Head First: I have to ask, though, I heard some of your new stuff is for building web apps. How does that relate?

HTML5: The biggest thing is that I'm not just for making web *pages* anymore; I'm designed for making full-blown web *applications*.

Head First: What's the difference?

HTML5: Web pages are mostly static pages. You'll have some images and a bunch of links, and a few nice effects here and there, like on the menus, but for the most part pages are for *reading*. Web applications, on the other hand, are for *interacting* with, *doing* stuff with. Like the applications on your desktop, only with web applications, you're doing stuff on the Web.

Head First: Can you give me an example?

HTML5: Social media apps, mapping apps, games...the list is endless.

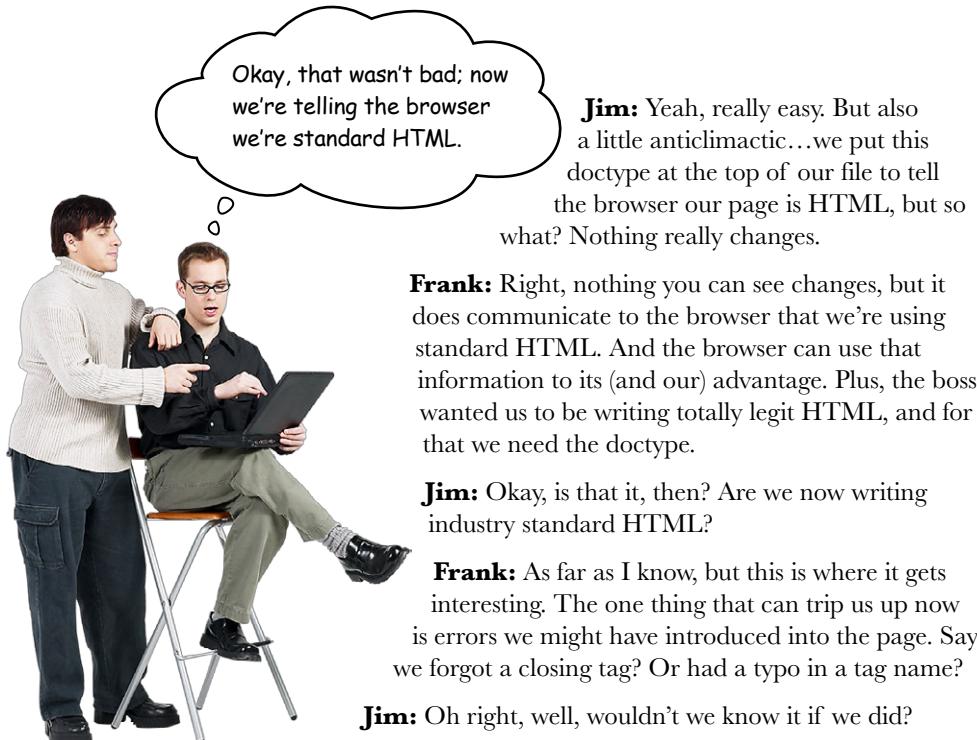
Head First: We couldn't do that stuff before HTML5?

HTML5: Well, you could do some of it, but a lot of the features required to build those kinds of applications are being standardized for the first time with me. Before, if they existed at all, they were somewhat haphazard.

Head First: I don't think we're going to be building any apps in this book, though.

HTML5: No, but check out *Head First HTML Programming*. That book is all about building web applications with me!

Head First: We will! Thanks for being here, HTML5.



Jim: Yeah, really easy. But also a little anticlimactic...we put this doctype at the top of our file to tell the browser our page is HTML, but so what? Nothing really changes.

Frank: Right, nothing you can see changes, but it does communicate to the browser that we're using standard HTML. And the browser can use that information to its (and our) advantage. Plus, the boss wanted us to be writing totally legit HTML, and for that we need the doctype.

Jim: Okay, is that it, then? Are we now writing industry standard HTML?

Frank: As far as I know, but this is where it gets interesting. The one thing that can trip us up now is errors we might have introduced into the page. Say we forgot a closing tag? Or had a typo in a tag name?

Jim: Oh right, well, wouldn't we know it if we did?

Frank: Not necessarily; the browser is pretty good at winging it when it sees errors.

Jim: How about I get the guys together, and we do a review of the entire page?

Frank: You may not need to...there are tools out there to help validate the page.

Jim: Validate?

Frank: Right, to go through the page and make sure all the markup is valid. Make sure we're keeping to the standard. It's a bit like a spell checker for HTML.

Jim: Sounds like a good idea. Where do we get these tools?

Frank: The standards guys over at the W3C have a validator, and it's free.

Jim: Great, let's do it.

Meet the W3C validator

Let's check out the W3C validator and have it validate our lounge files. To follow along, just point your browser to <http://validator.w3.org>.

The W3C validator is located at <http://validator.w3.org>.

There are three ways you can check your HTML:

- (1) If your page is on the Web, then you can type in the URL here and click the Check button, and the service will retrieve your HTML and check it.
- (2) You can choose the second tab, and upload a file from your computer. When you've selected the file, click Check, and the browser will upload the page for the service to check.
- (3) Or, choose the third tab, and copy and paste your HTML into the form on that tab. Then click Check and the service will check your HTML.

This validator checks the [markup validity](#) of Web documents in HTML, XHTML, SMIL, MathML, etc. If you wish to validate specific content such as [RSS/Atom feeds](#) or [CSS stylesheets](#), [MobileOK content](#), or to [find broken links](#), there are [other validators and tools](#) available.

The W3C validators are developed with assistance from the Mozilla Foundation, and supported by community donations. [Donate](#) and help us build better tools for a better web.

[VALIDATOR](#)

Validating the Head First Lounge

We're going to use the third tab, "Validate by Direct Input" to validate the "lounge.html" file. That means we need to copy and paste the HTML from "lounge.html" into the form on that tab; keep following along and give it a try...



The W3C Markup Validation Service

Check the markup (HTML, XHTML, ...) of Web documents

Validate by URI Validate by File Upload Validate by Direct Input

Validate by direct input

Enter the Markup to validate:

```
<!doctype html>
<html>
  <head>
    <title>Head First Lounge</title>
  </head>
  <body>
    <h1>Welcome to the New and Improved Head First Lounge</h1>
    
    <p>
      Join us any evening for refreshing
      <a href="elixir.html">elixirs</a>, conversation and
      maybe a game or two of <em>Dance Dance Revolution</em>.
    </p>
  </body>
</html>
```

More Options

Check

This validator checks the [markup validity](#) of Web documents in HTML, XHTML, SMIL, MathML, etc. If you wish to validate specific content such as [RSS/Atom feeds](#) or [CSS stylesheets](#), [MobileOK content](#), or to [find broken links](#), there are [other validators and tools](#) available.

The W3C validators are developed with assistance from the Mozilla Foundation, and supported by community donations. [Donate](#) and help us build better tools for a better web.

2931 Flattr

Home About... News Docs Help & FAQ Feedback Contribute

This service runs the W3C Markup Validator, v1.2.

COPYRIGHT © 1994-2010 W3C® (MIT, ERCIM, KEIO), ALL RIGHTS RESERVED. W3C LIABILITY, TRADEMARK, DOCUMENT USE AND SOFTWARE LICENSING RULES APPLY. YOUR INTERACTIONS WITH THIS SITE ARE IN ACCORDANCE WITH OUR PUBLIC AND MEMBER PRIVACY STATEMENTS.

VALIDATOR

We're using method (3) here. We clicked on the "Validate by Direct Input" tab and pasted the code for "lounge.html", which now has the doctype for HTML5 at the top, into the form. We're ready for the big moment...will the web page validate? Bets anyone? Click Check (and turn the page) to find out...

Feel free to use method (1) or (2) if it's more convenient.

Houston, we have a problem...

That red on the page can't be good. It doesn't look like the page validated. We'd better take a look...

We failed the validation. It looks like there is one error.

Result: 1 Error, 3 warning(s)

```
<!doctype html>
<html>
<head>
<title>Head First Lounge</title>
</head>
<body>
<h1>Welcome to the New and Improved Head First Lounge</h1>

<p>Join us any evening for refreshing
<a href="elixir.html">elixirs</a>, conversation and
maybe a game or two of <em>Dance Dance Revolution</em>.
```

Encoding: utf-8 **Doctype:** HTML5 **Root Element:** html

This must be the error.

Validation Output: 1 Error

Line 8, Column 29: An img element must have an alt attribute, except under certain conditions. For details, consult guidance on providing text alternatives for images.

```

```



Watch it!

The W3C is constantly revising the validator.

Because the W3C frequently revises the validator, you may not see exactly the same error messages. No worries, just keep following along because all the stuff in the next few pages is important, even if you don't see the error above.

This doesn't look bad. It looks like we have to use the alt attribute in the `` element.

Fixing that error

Okay, this looks pretty simple to fix. You just need to add an alt attribute to your `` elements in HTML5. Go ahead and open “lounge.html”, make the change, save, and then let’s try to validate again.

```
<!doctype html>
<html>
  <head>
    <title>Head First Lounge</title>
  </head>
  <body>
    <h1>Welcome to the New and Improved Head First Lounge</h1>
     ← You know the alt attribute; add it into the <img> element.
    <p>
      Join us any evening for refreshing
      <a href="elixir.html">elixirs</a>, conversation and
      maybe a game or two of <em>Dance Dance Revolution</em>.
      Wireless access is always provided; BYOWS (Bring
      your own web server).
    </p>
    <h2>Directions</h2>
    <p>
      You'll find us right in the center of downtown
      Webville. If you need help finding us, check out our
      <a href="directions.html">detailed directions</a>.
      Come join us!
    </p>
  </body>
</html>
```



Why do you think the alt attribute is required in HTML5?

We're almost there...

Success! We have a green bar on the page; that must be good. But there are three warnings. That sounds like we've still got a few things to take care of. Let's take a look:

The screenshot shows the W3C Markup Validation Service interface. At the top, it says "This document was successfully checked as HTML5!". Below that, under "Result:", it says "Passed, 3 warning(s)". The "Source:" section contains the HTML code for the "Head First Lounge" page. Under "Encoding:", "Doctype:", and "Root Element:", the values "utf-8", "HTML5", and "html" are listed respectively.

Notes and Potential Issues

The following notes and warnings highlight missing or conflicting information which caused the validator to perform some guesswork prior to validation, or other things affecting the output below. If the guess or fallback is incorrect, it could make validation results entirely incoherent. It is *highly recommended* to check these potential issues, and, if necessary, fix them and re-validate the document.

Using experimental feature: **HTML5 Conformance Checker**. The validator checked your document with an experimental feature: *HTML5 Conformance Checker*. This feature has been made available for your convenience, but be aware that it may be unreliable, or not perfectly up to date with the latest development of some cutting-edge technologies. If you find any issues with this feature, please [report them](#). Thank you.

No Character encoding declared at document level. No character encoding information was found within the document, either in an HTML `meta` element or an XML declaration. It is often recommended to declare the character encoding in the document itself, especially if there is a chance that the document will be read from or saved to disk, CD, etc. See [this tutorial on character encoding](#) for techniques and explanations.

Using Direct Input mode: UTF-8 character encoding assumed. Unlike the "by URL" and "by File Upload" modes, the "Direct Input" mode of the validator provides validated content in the form of characters pasted or typed in the validator's form field. This will automatically make the data UTF-8, and therefore the validator does not need to determine the character encoding of your document, and will ignore any charset information specified. If you notice a discrepancy in detected character encoding between the "Direct Input" mode and other validator modes, this is likely to be the reason. It is neither a bug in the validator, nor in your document.

We passed! But...

...there are a few warnings; we should scroll down and take a look...

No worries here—this is a standard warning you'll always see as long as the validator is considered to be experimental by the W3C (which could be for a long time).

Hmm, this looks like a problem caused by leaving out some information about your character encoding. We'll see what that's about in a sec...

And this just says that they are going to assume a character encoding, given we didn't supply one.

So, we've got a valid file in terms of how we've written the HTML, but it looks like we need to do something about our "character encoding." Let's take a look at what that means...



No Character encoding declared at document level

No character encoding information was found within the document, either in an HTML `meta` element or an XML declaration. It is often recommended to declare the character encoding in the document itself, especially if there is a chance that the document will be read from or saved to disk, CD, etc.

Frank: The character encoding tells the browser what kind of characters are being used in the page. For instance, pages can be written using encodings for English, Chinese, Arabic, and lots of other types of characters.

Jim: What's so hard about figuring out how to display a character? If there's an "a" in the file, then the browser should display an "a". Right?

Frank: Well, what if you're using Chinese in your pages? It's an entirely different "alphabet" and it has a heck of a lot more than 26 A-Z characters.

Jim: Oh. Good point...but shouldn't the browser be able to tell the difference? Those other languages look nothing like English.

Frank: No, the browser is just reading data. It can try to guess what kind of character encoding to use, but what if it's wrong? This can lead not only to pages being displayed wrong, but also potential exploits from hackers. The character encoding takes the guesswork out of it.

Jim: We've had the site up for a long time. Why is this an issue now?

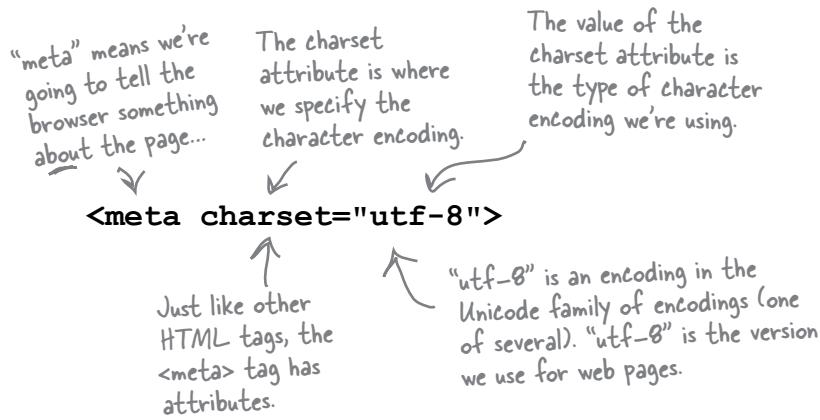
Frank: Because the validator is saying "Hey, if I'm going to validate your page, you'd better tell me up front what characters you're going to use!" And think about it, we'd want to do that for the browsers out there anyway. Don't stress, we just need to add one more line to our HTML, called a `<meta>` tag. I should have thought of this sooner.

Jim: Got any other surprises for us? I really thought our web page would validate after we put the document type definition in our file...

Frank: I sure hope there are no more surprises! Let's get the `<meta>` tag in there and find out.

Adding a `<meta>` tag to specify the character encoding

Character encodings give us a way to represent all the letters, numbers and other symbols in our language on the computer. You might know of some of these encodings, like ASCII or even Morse code, and there are many other encodings out there. Luckily, the world has now standardized on the Unicode character encoding. With Unicode, we can represent all languages with one type of encoding. But, given there are other encodings out there, we still need to tell the browser we're using Unicode (or another encoding of your choice). To specify Unicode for your web pages, you'll need a `<meta>` tag in your HTML that looks like this:



there are no Dumb Questions

Q: Doctypes, `<meta>` tags...ugh, do I need to really do all this to write web pages?

A: Specifying a doctype and character encoding with a `<meta>` tag are kind of like taxes: you gotta do them to be compliant. Look at it this way: you already understand them more than 98% of the web page writing population, which is great. But at the end of the day, everyone just puts the doctype and `<meta>` tag in their HTML and moves on with life. So make sure you've got them in your HTML, and then go do something much more fun.

Q: utf-8?

A: Work with us here. It's like WD-40; you don't worry about why it's called that, you just use it. As we said, utf-8 (also written sometimes as UTF-8) is part of the Unicode encoding family. The *u* in utf-8 means Unicode. Unicode is a character set supported across many commonly used software applications and operating systems, and is the encoding of choice for the Web, because it

supports all languages, and multilingual documents (documents that use more than one language). It's also compatible with ASCII, which was a common encoding for English-only documents. If you're interested in learning more about Unicode or character encodings in general, check out the information on character encoding at <http://www.w3.org/International/O-charset.html>.

Q: I've also seen `<meta>` tags that look like this: `<meta http-equiv="Content-Type" content="text/html; charset=utf-8" >`. Do I need to use this instead sometimes?

A: No. That is the format for the `<meta>` tag in HTML 4.01 and earlier. In HTML5, you can just write `<meta charset="utf-8">`.

Q: Is this why you had us save our files using utf-8 for the encoding way back in Chapter 1?

A: Yes. You want the encoding of the file you're serving to the browser to match the encoding you specify in the `<meta>` tag.

Making the validator (and more than a few browsers) happy with a <meta> tag...

The <meta> tag belongs in the <head> element (remember that the <head> contains information *about* your page). Go ahead and add the <meta> tag line right into your HTML. Let's first add it to the "lounge.html" file:

```
<!doctype html>          Here's the <meta> tag. We've
<html>                  added it to the <head> element
    <head>              above the <title> element.
        <meta charset="utf-8">      ↗   ↙ Add this line above any other
        <title>Head First Lounge</title> elements in the <head> element.
    </head>
    <body>
        <h1>Welcome to the New and Improved Head First Lounge</h1>
        
        <p>
            Join us any evening for refreshing
            <a href="elixir.html">elixirs</a>, conversation and
            maybe a game or two of <em>Dance Dance Revolution</em>.
            Wireless access is always provided; BYOWS (Bring
            your own web server).
        </p>
        <h2>Directions</h2>
        <p>
            You'll find us right in the center of downtown
            Webville. If you need help finding us, check out our
            <a href="directions.html">detailed directions</a>.
            Come join us!
        </p>
    </body>
</html>
```

Want to place another bet? Is this going to validate? First, make the changes to your "lounge.html" file, save it, and reload it into your browser. Once again, *you* won't notice any change, but the *browser* will. Now let's see if it validates...

Third time's the charm?

This time, we picked the second tab (validate by file upload). You can choose whichever method works best for you. If you want to try the upload method, then upload your “lounge.html” HTML file to the W3C validator web page at <http://validator.w3.org>. Once you’ve done that, click the Check button...

Successfully checked as HTML5!
Love the green!

We still have one warning...but we don't need to worry about it (see below).

Success! We can tell the boss we're writing totally industry standard HTML, and we can even say we're ready for HTML5.

O
O

This document was successfully checked as HTML5!

Result: Passed, 1 warning(s)

File: no file selected
Use the file selection box above if you wish to re-validate the uploaded file lounge.html

Encoding: utf-8

Doctype: HTML5

Root Element: html

I ❤ VALIDATOR The W3C validators rely on community support for hosting and development.
[Donate](#) and help us build better tools for a better web.

2936 [Feedback](#)

Options

Show Source Show Outline List Messages Sequentially Group Error Messages by Type
 Validate error pages Verbose Output Clean up Markup with HTML-Tidy

[Help](#) on the options is available. [Revalidate](#)

Notes and Potential Issues

The following notes and warnings highlight missing or conflicting information which caused the validator to perform some guesswork prior to validation, or other things affecting the output below. If the guess or fallback is incorrect, it could make validation results entirely incoherent. It is *highly recommended* to check these potential issues, and, if necessary, fix them and re-validate the document.

Using experimental feature: **HTML5 Conformance Checker**.

The validator checked your document with an experimental feature: **HTML5 Conformance Checker**. This feature has been made available for your convenience, but be aware that it may be unreliable, or not perfectly up to date with the latest development of some cutting-edge technologies. If you find any Issues with this feature, please [report them](#). Thank you.

Congratulations

The uploaded document "lounge.html" was successfully checked as HTML5. This means that the resource in question identified itself as "HTML5" and that we successfully performed a formal validation of it. The parser implementations we used for this check are based on [validator.nu \(HTML5\)](#).

This is just the same warning about the fact that we're using an "experimental service." Nothing to worry about.

^{there are no} Dumb Questions

Q: The validator says it is experimental for HTML5. What does that mean?

A: The message “Using experimental feature: HTML5 Conformance Checker” in the validator means that the validator is checking your HTML according to the HTML5 standard, but because the HTML5 standard isn’t final (and still has new features being added), the validator is prone to change, so the results you get when you validate your page aren’t set in stone. That means, as a conscientious developer, it’s in your best interest to stay up-to-date on the HTML standard, and check your pages fairly regularly.

Q: What have we really achieved in this chapter? My page still looks the same.

A: In this chapter we’ve tweaked your page slightly so that it is compliant with the HTML specification. What good is that? The closer you are to the spec, the more likely that your page is going to perform well in the real world. If you’re producing a professional web page, you want it to be written using the industry standard, and that’s what we’ve done in this chapter by adding a doctype, setting a character encoding, and cleaning up an oversight (the alt attribute) in the HTML.

Q: Why do we need that alt attribute anyway?

A: For two great reasons. First, if your image is broken for some reason (say, your image server goes down, or your connection is really slow), the alt attribute will (in most browsers) show the alt text you’ve specified in place of the image. Second, for vision-impaired users who are using a screen reader to read the page, the screen reader will read the alt text to the user, which helps them understand the page better.

Q: What if I tell the browser I’m using HTML5, and I’m not?

A: The browser will figure out that you’re not really writing HTML5 and use the error handling capabilities it has to try to do the right thing. And then you’re back to the problem of having the various browsers handle your page in different ways. The only way you can get predictable results is to tell the browser you’re using HTML5 and to actually do so, properly.

Q: We talked a little about HTML5, but I want make sure I’m clear: is there any difference between the HTML we’re writing and HTML5?

A: We’re using standard HTML, which is HTML5. Now, HTML5 introduced some new markup (which we’ll be seeing soon enough) as well as support for writing web applications (which we won’t be doing in this book), but HTML5 is HTML, and everything you’ve been writing is HTML5 “compliant.” So, sorry for the terminology, but going forward everything is just HTML, including all the new features provided by the HTML5 specification.

The good news is that everything you’ve learned is all ready for HTML5, and in fact you see how little you actually had to do to go from an “informal HTML” page to a professional HTML page. That said, you might want to tell your boss you’re already using HTML5 just for bonus points toward your next raise.

Q: What’s the big deal with HTML5 compared to HTML 4.01 anyway?

A: The big deal about HTML5 is threefold. First, there are some new elements and attributes in HTML5 that are pretty cool (like the <video> element), and others that will help you write better pages (we’ll be getting to those later in the book).

Second, there are many new features that allow web developers to create *web applications* with HTML5. Web applications are web pages that behave more like applications (like the ones you’re used to using on your laptop or mobile device) than static web pages. If you’re interested in creating web applications, then after you’re all done with this book (cue shameless plug), you should check out *Head First HTML5 Programming* (O’Reilly).

Finally, the HTML5 specification is a lot more robust than the specifications for the previous versions of HTML. Remember how we said that the specification is now documenting common errors that web developers make? And helping browsers to know how to handle those errors? That means that web pages with errors on them don’t cause the havoc they used to, which is a good thing for users.

All in all, HTML5 is a big improvement over HTML 4.01, and well worth learning. We’ll get you up to speed quickly over the next few chapters.



Your turn. Add the `<meta>` tag to “directions.html” and “elixir.html”. Try validating them; do they validate? If not, fix them so that they do.

Exercise

Use this space for notes about your validating experience! ↗



Time to play a little game with the validator. Take the code you just successfully validated as HTML5 (on page 241), and remove the doctype. That's right—remove it, just to see what happens when you validate. Go ahead and submit this version of the file to the validator and see what happens. Make notes below about the errors you get.

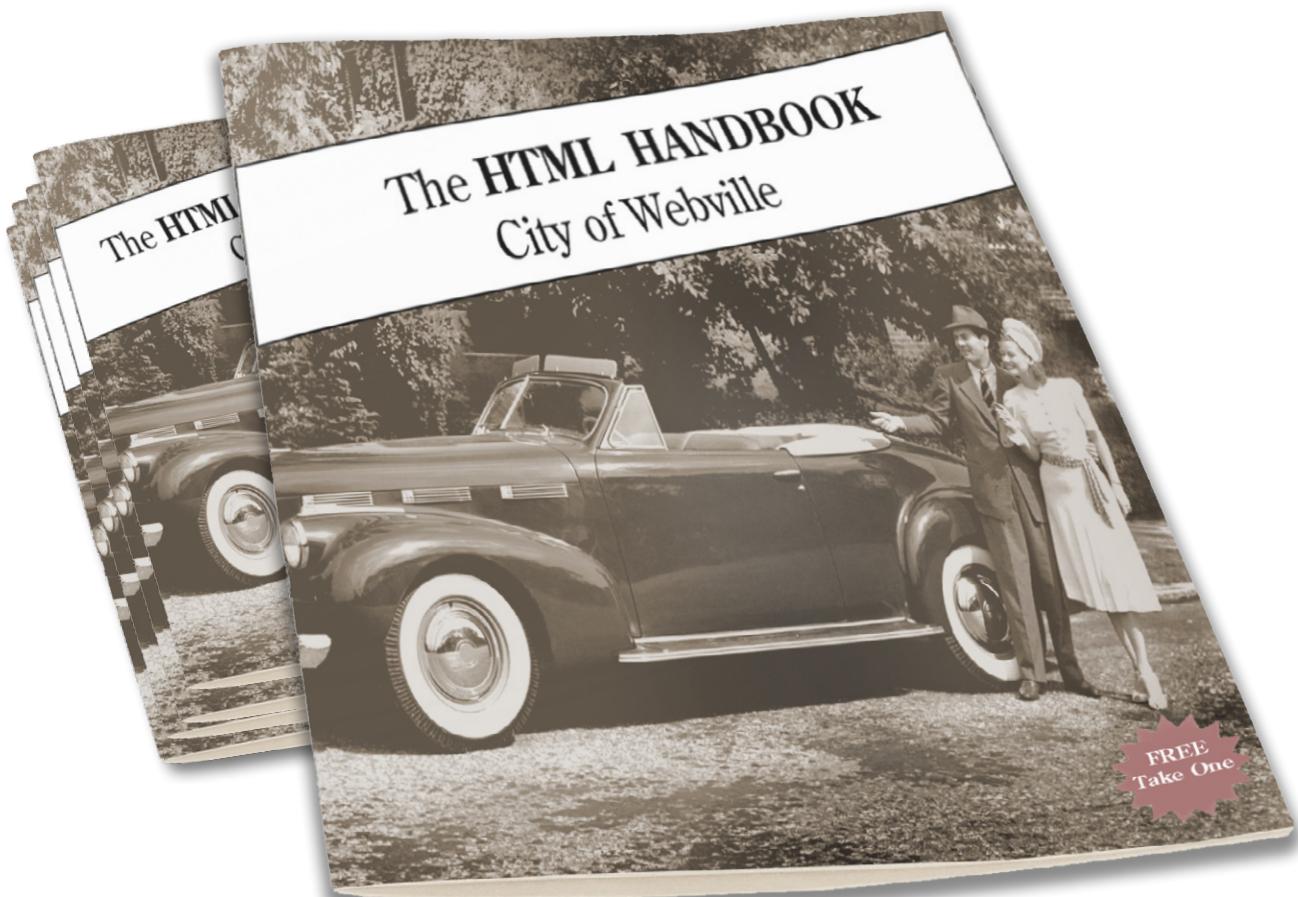
```
<!doctype html> ↙ Remove the doctype!
<html>
  <head>
    <meta charset="utf-8">
    <title>Head First Lounge</title>
  </head>
  <body>
    <h1>Welcome to the New and Improved Head First Lounge</h1>
    
    <p>
      Join us any evening for refreshing
      <a href="elixir.html">elixirs</a>, conversation and
      maybe a game or two of <em>Dance Dance Revolution</em>.
      Wireless access is always provided; BYOWS (Bring
      your own web server).
    </p>
    <h2>Directions</h2>
    <p>
      You'll find us right in the center of downtown
      Webville. If you need help finding us, check out our
      <a href="directions.html">detailed directions</a>.
      Come join us!
    </p>
  </body>
</html>
```

Your notes here. How many ↗
errors did you get?

What does this tell you about the type of
your HTML if you don't include a doctype?

Calling all HTML professionals, grab the handbook...

Welcome to the elite set of HTML crafters, those who know how to create professional pages. There's a lot to remember, so the City of Webville prepared a handy guide to creating industry standard pages. This guide is meant for you—someone who is new to Webville. It isn't an exhaustive reference, but rather focuses on the more important best practices in building your pages. And you'll definitely be adding to the knowledge in this guide as you get to know your way around Webville in coming chapters. But for now, take one—they're FREE.



Webville Guide to HTML

In this handy guide, we've boiled down writing well-formed HTML pages into a common sense set of guidelines. Check them out:

Always begin with the <doctype>.



Always start each page with a doctype. This will get you off on the right foot with browsers, and with the validator too.

Use <!doctype html> at all times, unless you really are writing HTML 4.01 or XHTML.



The <html> element: don't leave home without it.

Following the doctype, the <html> element must always be the top, or root, element of your web page. So, after the doctype, the <html> tag will start your page and the </html> tag should end it, with everything else in your page nested inside.



Remember to use both your <head> and your <body> for better HTML.

Only the <head> and <body> elements can go directly inside your <html> element. This means that every other element must go either inside the <head> or the <body> element. No exceptions!



Feed your <head> the right character encoding.

Include a <meta charset="utf-8"> tag in your <head>. The browser will thank you, and so will your users when they're reading comments on your blog from users around the world.

Webville Guide to HTML, continued

In this handy guide, we've boiled down writing well-formed HTML pages into a common sense set of guidelines. Check them out:



What's a `<head>` without a `<title>`?

Always give your `<head>` element a `<title>` element. It's the law. Failure to do so will result in HTML that isn't compliant. The `<head>` element is the only place you should put your `<title>`, `<meta>`, and `<style>` elements.



Be careful about nesting certain elements.

Within the guidelines we've provided here, the nesting rules are fairly flexible. But there are a couple of cases that don't make sense. Never nest an `<a>` element inside another `<a>` element because that would be too confusing for our visitors. Also, void elements like `` provide no way to nest other inline elements within them.



Check your attributes!

Some element attributes are required, and some are optional. For instance, the `` element wouldn't make much sense without a `src` attribute, and now you know the `alt` attribute is required too. Get familiar with the required and optional attributes of each element as you learn it.



HTML Archaeology



Throughout this book you've been using elements and attributes that are all part of the current HTML standard. So, you haven't had much opportunity to see the phased-out elements and attributes. Most of those elements actually got phased out in HTML 4.01, but they may still be hanging around in old web pages, so it doesn't hurt to know a little about these legacy elements. We did some digging and found an HTML 3.2 page that contains some elements and attributes that are no longer part of the standard, as well as a couple of common mistakes that are not recommended in modern HTML.

```

<html>
  <head>
    <title>Webville Forecast</title>
  </head>
  <body bgcolor="tan" text="black">

    <p>
      The weather report says lots of rain and wind in store for
      <font face="arial">Webville</font> today, so be sure to
      stay inside if you can.
    </p>
    <ul>
      <li>Tuesday: Rain and 60 degrees. <!--
      <li>Wednesday: Rain and 62 degrees. <!--
    </ul>

    <p align=right>
      Bring your umbrella!
    </p>

    <center><font size="small">This page brought to you by Lou's
      Diner, a Webville institution for over 50 years.
    </font></center>
  </body>
</html>

```

Here are some attributes that controlled presentation. bgcolor sets the background color of the page, and text sets the color of the body text.

Font changes were made with the `` element and its face attribute.

You could get away without some closing tags, like `` and `</p>`. You sometimes still can, but it is not recommended!!

Missing quotes around attribute values. Quotes are always recommended now, and required for attributes with multiple values.

Here were two ways to align text. Right-align a paragraph, or center a piece of text.

Text size was controlled with the `` element, using the size attribute.



BE the Validator

Below, you'll find an HTML file. Your job is to play like you're the validator and locate ALL the errors. After you've done the exercise, look at the end of the chapter to see if you caught them all.

Use the validator to check your work once you're done (or if you need hints).

```
<html>
<head>
    <meta charset="utf-9">
</head>
<body>
    
    <h1>Tips for Enjoying Your Visit in Webville
    <p>
        Here are a few tips to help you better enjoy your stay in Webville.
    </p>
    <ul>
        <li>Always dress in layers and keep an html around your
            head and body.</li>
        <li>Get plenty of rest while you're here, sleep helps all
            those rules sink in.</li>
        <li>Don't miss the work of our local artists right downtown
            in the CSS gallery.
    </ul>
    </p>
    <p>
        Having problems? You can always find answers at
        <a href="http://wickedlysmart.com"><em>WickedlySmart</em></a>.
        Still got problems? Relax, Webville's a friendly place, just ask someone
        for help. And, as a local used to say:
    </p>
    <em><p>
        Don't worry. As long as you hit that wire with the connecting hook
        at precisely 88mph the instant the lightning strikes the tower...
        everything will be fine.
    </em></p>
</body>
</html>
```



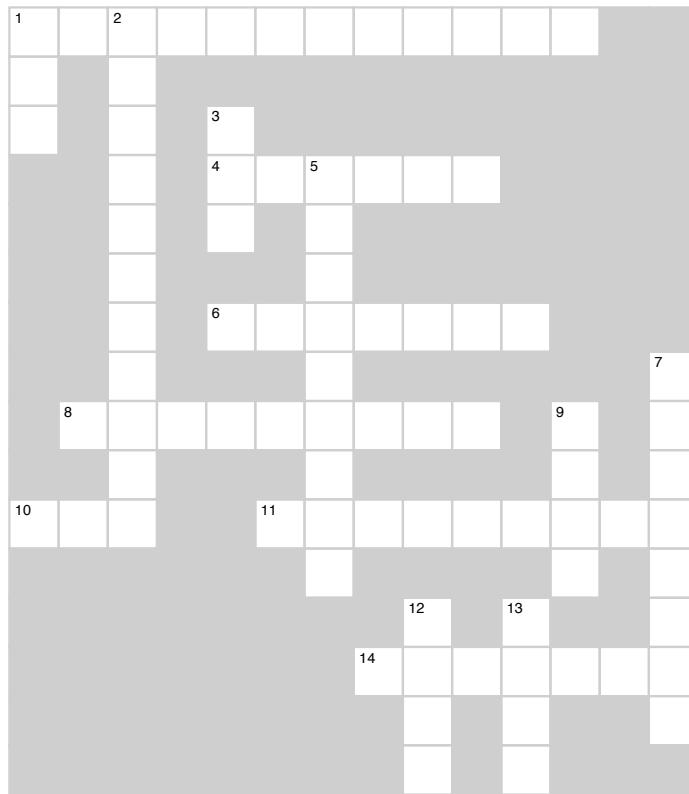
BULLET POINTS

- HTML5 is the current HTML standard.
- The World Wide Web Consortium (W3C) is the standards organization that defines what standard HTML is.
- The document type definition (doctype) is used to tell the browser the version of HTML you're using.
- The HTML standard is now a "living standard," which means that the standard will change to incorporate new features and updates.
- The `<meta>` tag in the `<head>` element tells the browser additional information about a web page, such as the content type and character encoding.
- The `charset` attribute of the `<meta>` tag tells the browser the character encoding that is used for the web page.
- Most web pages use the utf-8 encoding for HTML files, and for the `<meta>` tag `charset` attribute.
- The `alt` attribute is required for the `` element.
- The W3C validator is a free online service that checks pages for compliance with standards.
- Use the validator to ensure that your HTML is well formed and that your elements and attributes meet the standard.
- By adhering to the standard, your pages will display more quickly and with fewer display differences between browsers, and your CSS will work better.



HTMLcross

It's been a heck of a chapter. Go ahead and grab a cup of your favorite beverage, sit back, and strengthen those neural connections by doing this crossword. All the answers come from the chapter.



Across

1. Victim of the browser wars.
4. The HTML standard is a _____ standard.
6. Required in the <head> element.
8. Web standards makers have promised future HTML will be _____ compatible with older HTML.
10. The boss wanted to standardize before adding _____ to the Lounge pages.
11. When your HTML meets the standard, it is this.
14. Definition that tells the browser and validator what kind of document you're creating.

Down

1. Standards organization that supplies the validator.
2. Microsoft versus Netscape.
3. attribute required in standard HTML.
5. This service will check your HTML for compliance with the standard.
7. The older _____ were much more complicated compared to the newest one.
9. Where you put information about the page.
12. Where you put web page content.
13. The most common encoding for web pages.

BE the Validator Solution



Below, you'll find an HTML file. Your job is to play like you're the validator and locate ALL the errors. Here's the solution.

```

Missing doctype
<html>
<head>
  <meta charset="utf-9"> Should be "utf-8" instead of
                                "utf-9" (which doesn't exist!)
  <title> should be
        in the <head>.
</head>
<body>
   No alt attribute
  <h1>Tips for Enjoying Your Visit in Webville <!-- Missing </h1> tag. This will cause
  <p> problems with the <p> element below.
    Here are a few tips to help you better enjoy your stay in Webville. <!--
  </p>
  <ul>
    <li>Always dress in layers and keep an html around your
        head and body.</li>
    <li>Get plenty of rest while you're here, sleep helps all
        those rules sink in.</li>
    <li>Don't miss the work of our local artists right downtown
        in the CSS gallery. <!-- Missing </li> tag. This will still validate,
        but it's not recommended!
  </ul>
  </p> <!-- Extra </p> that doesn't match a <p>
  <p>
    Having problems? You can always find answers at
    <a href="http://wickedlysmart.com"><em>WickedlySmart</em></a>.
    Still got problems? Relax, Webville's a friendly place, just ask someone
    for help. And, as a local used to say:
  </p>
  <em><p> <!-- <em> and <p> tags are switched.
    Don't worry. As long as you hit that wire with the connecting hook
    at precisely 88mph the instant the lightning strikes the tower...
    everything will be fine.
  </em></p>
</body>
</html>
```



HTMLcross Solution

¹ W	E	² B	D	E	V	E	L	O	P	E	R	
3		R										
C	O		³ A									
	W		⁴ L	I	⁵ V	I	N	G				
S	T		T	A								
E				L								
R			⁶ <	T	I	T	L	E	>			
W				D						⁷ D		
⁸ B	A	C	K	W	A	R	D	S	⁹ H	O		
R				T					E	C		
¹⁰ C	S	S		¹¹ C	O	M	P	L	I	A	N	T
				R					D		Y	
					¹² B		¹³ U				P	
					D	O	C	T	Y	P	E	
					D		F				S	
					Y		8					



Your turn. Add the strict doctype and the `<meta>` tag to “directions.html” and “elixir.html”. Try validating them—do they validate? If not, fix them so that they do.

Solution: To validate “elixir.html”, you’ll have to add the `alt` attribute to each of your `` elements.



Time to play a little game with the validator. Take the code you just successfully validated as HTML5 (on page 241), and remove the doctype. That's right—remove it, just to see what happens when you validate. Go ahead and submit this version of the file to the validator and see what happens. Make notes below about the errors you get.

```
<!doctype html> ↵ Remove the doctype!
<html>
  <head>
    <meta charset="utf-8">
    <title>Head First Lounge</title>
  </head>
  <body>
    <h1>Welcome to the New and Improved Head First Lounge</h1>
    
    <p>
      Join us any evening for refreshing
      <a href="elixir.html">elixirs</a>, conversation and
      maybe a game or two of <em>Dance Dance Revolution</em>.
      Wireless access is always provided; BYOWS (Bring
      your own web server).
    </p>
    <h2>Directions</h2>
    <p>
      You'll find us right in the center of downtown
      Webville. If you need help finding us, check out our
      <a href="directions.html">detailed directions</a>.
      Come join us!
    </p>
  </body>
</html>
```

We get three errors and four warnings if we try to validate without the doctype. The validator assumes we're writing HTML 4.01 Transitional (which was a version of HTML 4.01 designed to use while you were "transitioning" to XHTML). The validator really doesn't like that there's no doctype, and complains a couple of times about that. It also complains about the `<meta charset="utf-8">`, because before HTML5, charset was not a valid attribute of the `<meta>` tag. You can get the idea that using a doctype makes both the validator, and the browsers, happier campers.

Your notes here. How many ↗
errors did you get?

7 getting started with CSS

Adding a Little Style



I was told there'd be CSS in this book. So far you've been concentrating on learning HTML to create the structure of your web pages. But as you can see, the browser's idea of style leaves a lot to be desired. Sure, we could call the fashion police, but we don't need to. With CSS, you're going to completely control the presentation of your pages, often without even changing your HTML. Could it really be so easy? Well, you *are* going to have to learn a new language; after all, Webville is a bilingual town. After reading this chapter's guide to learning the language of CSS, you're going to be able to stand on *either* side of Main Street and hold a conversation.

You're not in Kansas anymore

You've been a good sport learning about markup and structure and validation and proper syntax and nesting and compliance, but now you get to really start *having some fun* by styling your pages. But no worries, all those HTML push-ups you've been doing aren't going to waste. In fact, you're going to see that a solid understanding of HTML is crucial to learning (and using) CSS. And learning CSS is just what we're going to do over the next several chapters.

Just to tease you a bit, on these two pages we've sprinkled a few of the designs you're going to work with in the rest of the book. Quite a difference from the pages you've been creating so far, isn't it? So, what do you need to do to create them? Learn the language of CSS, of course.

Let's get started...

The image displays three screenshots of web pages from Head First Lounge and Starbuzz Coffee websites. The top right screenshot shows a page titled "Our Elixirs" featuring a green tea cooler. The middle right screenshot shows a page titled "Head First Lounge" with a large martini glass graphic and a welcome message. The bottom left screenshot shows a form titled "The Starbuzz Bean Machine" for ordering beans, with fields for bean type, number of bags, arrival date, extras, and shipping information. To the right of the form, there are sections for "BEVERAGES", "ELIXIRS", and "Starbuzz meets social media" and "Starbuzz uses computer science". The bottom right screenshot shows a blog post from Starbuzz Coffee with a headline about meeting social media.

Remember the Wizard of Oz? Well, this is the part of the book where things go from black and white to color.

Our Elixirs

Green Tea Cooler

Head First Lounge

Welcome to the Head First Lounge

The Head First Lounge is, no doubt, the biggest trendsetter in Webville. Stop in to sample the eclectic offering of elixirs, teas, and coffees, or, stay a bit longer and enjoy the multicultural culinary menu that combines a harmony of taste, texture, and color with the beat in fresh and healthy ingredients.

During your stay at the lounge, you'll enjoy a smooth mixture of ambient and mystic sounds, filling the lounge with sights from eras past. And, don't forget, the lounge offers free wireless access to the Internet, so bring your laptop.

Our guarantee: at the lounge, we're committed to providing you, our guests, with the best service and atmosphere.

Our guarantee: at the lounge, we're committed to providing you, our guests, with the best service and atmosphere.

Starbuzz Coffee - Blog

file:///chapter12/starbuzz/blog.html

Starbuzz meets social media

Starbuzz uses computer science

Most unique patron of the month

Our most unique patron of the month award goes to a customer in Poulsbo, Washington, whose daily morning order is a "six-splenda, no-foam, 130-degree non-fat soy latte, with the splenda stirred in before the milk is added." Do we have unique customers or what?

© 2012, Starbuzz Coffee. All trademarks and registered trademarks appearing on this site are the property of their respective owners.

The Starbuzz Bean Machine

Fill out the form below and click "order now" to order.

Choose your beans: House Blend

Type: Whole bean Ground

Number of bags: 2

Must arrive by date: 5/12/2012

Extras: Gift wrap Include catalog with order

Ship to:

Name: Buckaroo Banzai

Address: Banzai Institute

City: Los Angeles

State: CA

Zip: 90050

Phone: 310-555-1212

Customer Comments: Great coffee!

Order Now

Overheard on Webville's "Trading Spaces"

Not up on the latest reality TV? No problem, here's a recap: take two neighbors, two homes, and \$1,000. The two neighbors switch homes, and using the \$1,000, totally redesign a room or two in 48 hours. Let's listen in...



Of course, in the Webville edition of the show, everyone talks about design in CSS. If you're having trouble understanding them, here's a little translation tip: each statement in CSS consists of a location (like `bedroom`), a property in that location (like `drapes` or `carpet`), and a style to apply to that property (like the color blue, or 1 inch tiles).

Using CSS with HTML

We're sure CSS has a bright future in the home design category, but let's get back to HTML. HTML doesn't have rooms, but it does have elements, and those elements are going to be the locations that we're styling. Want to paint the walls of your `<p>` elements red? No problem; only paragraphs don't have walls, so you're going to have to settle for the paragraph's `background-color` property instead. Here's how you do that:

The first thing you do is select the element you want to style, in this case the `<p>` element. Notice in CSS, you don't put `<>` around the name.

```
p {  
    background-color: red;  
}
```

Place all the styles for the `<p>` element in between `{ }` braces.

Then you specify the property you want to style, in this case the `<p>` element's `background-color`.

And you're going to set the `background-color` to red.

At the end, put a semicolon.

There's a colon in between the property and its value.

We call the whole thing a RULE.

You could also write the rule like this:

```
p { background-color: red; }
```

Here, all we've done is remove the linebreaks. As with HTML, you can format your CSS pretty much as you like. For longer rules, you'll usually want to add some linebreaks and indenting to make the CSS more readable (for you).

Wanna add more style?

You can add as many properties and values as you like in each CSS rule. Say you wanted to put a border around your paragraphs, too. Here's how you do that:

```
p {  
    background-color: red;  
    border: 1px solid gray;  
}
```

The `<p>` element will have a border...

All you have to do is add another property and value.

...that is 1 pixel thick, solid, and gray.

^{there are no} Dumb Questions

Q: Does every `<p>` element have the same style? Or can I, say, make two paragraphs different colors?

A: The CSS rules we've used so far define the style for all paragraphs, but CSS is very expressive: it can be used to specify styles in lots of different ways, for lots of different elements—even subsets of elements. You'll see how to make paragraphs two different colors later in this chapter.

Q: How do I know what properties I can set on an element?

A: Well, there are lots of properties that can be set on elements, certainly more than you'd want to memorize, in any case. You're going to get quite familiar with the more common properties in the next few chapters. You'll probably also want to find a good CSS reference. There are plenty of references online, and O'Reilly's *CSS Pocket Reference* is a great little book.

Q: Remind me why I'm defining all this style in a separate language, rather than in HTML. Since the elements are written in HTML, wouldn't it be easier just to write style in HTML, too?

A: You're going to start to see some big advantages to using CSS in the next few chapters. But here's a quick answer: CSS really is better suited for specifying style information than HTML. Using just a small bit of CSS, you can create fairly large effects on the style of your HTML. You're also going to see that CSS is a much better way to handle styles for multiple pages. You'll see how that works later in this chapter.



Say you have an `` element inside a paragraph. If you change the background color of the paragraph, do you think you also have to change the background of the `` element so it matches the background color of the paragraph?

Getting CSS into your HTML

You know a little about CSS syntax now. You know how to select an element and then write a rule with properties and values inside it. But you still need to get this CSS into some HTML. First, we need some HTML to put it in. In the next few chapters, we're going to revisit our old friends—Starbuzz, and Tony and his Segway journal—and make things a little more stylish. But who do you think is dying to have their site styled first? Of course, the Head First Lounge guys. So, here's the HTML for the Head First Lounge main page. Remember, in the last chapter we fixed things up a little and made it proper HTML (would you have expected any less of us?). Now, we're adding some style tags, the easiest way to get style into your pages.

But not necessarily the best way. We'll come back to this later in the chapter and see another way.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Head First Lounge</title>
    <style>
      Here's what we're interested in: the <style> element.
    </style>
    To add CSS style directly to your HTML, add
    opening and closing style tags in the <head> element.
  </head>
  <body>
    <h1>Welcome to the Head First Lounge</h1>
    <p>
      
    </p>
    <p>
      Join us any evening for refreshing
      <a href="#">beverages/elixir.html>elixirs</a>,
      conversation and maybe a game or two of
      <em>Dance Dance Revolution</em>.
      Wireless access is always provided;
      BYOWS (Bring your own web server).
    </p>
    <h2>Directions</h2>
    <p>
      You'll find us right in the center of downtown
      Webville. If you need help finding us, check out our
      <a href="#">about/directions.html>detailed directions</a>.
      Come join us!
    </p>
  </body>
</html>
```



Adding style to the lounge

Now that you've got the `<style>` element in your HTML, you're going to add some style to the lounge to get a feel for writing CSS. This design probably won't win you any "design awards," but you gotta start somewhere.

The first thing we're going to do is change the color (something to match those red lounge couches) of the text in the paragraphs. To do that, we'll use the CSS `color` property like this:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Head First Lounge</title>
    <style>
      Here's the rule that
      is going to specify
      the font color of the
      paragraphs.
      We're selecting just
      the <p> element to
      apply this style to.
      p {
        color: maroon;
      }
      We're setting the text to a
      lovely maroon color that happens
      to match the lounge couches.
    </style>
  </head>
  <body>
    <h1>Welcome to the Head First Lounge</h1>
    <p>
      
    </p>
    <p>
      Join us any evening for refreshing
      <a href="beverages/elixir.html">elixirs</a>,
      conversation and maybe a game or two of
      <em>Dance Dance Revolution</em>.
      Wireless access is always provided;
      BYOWS (Bring your own web server).
    </p>
    <h2>Directions</h2>
    <p>
      You'll find us right in the center of downtown
      Webville. If you need help finding us, check out our
      <a href="about/directions.html">detailed directions</a>.
      Come join us!
    </p>
  </body>
</html>
```

The diagram shows several annotations with arrows pointing to specific parts of the CSS code:

- An arrow points from the text "Here's the rule that is going to specify the font color of the paragraphs." to the `p {` selector.
- An arrow points from the text "We're selecting just the `<p>` element to apply this style to." to the `<p>` selector.
- An arrow points from the text "The property to change the font color is named "color" (you might think it would be "font-color" or "text-color", but it's not)." to the `color: maroon;` declaration.
- An arrow points from the text "We're setting the text to a lovely maroon color that happens to match the lounge couches." to the `color: maroon;` declaration.
- An arrow points from the text "The p selector selects all the paragraphs in the HTML." to the `p {` selector.

Cruising with style: the test drive

Go ahead and make all the changes from the last couple of pages to your “lounge.html” file in the “chapter7/lounge” folder, save, and reload the page in your browser. You’ll see that the paragraph text color has changed to maroon:



Instead of setting the color, what if you set background-color of the `<p>` elements to maroon instead? How would it change the way the browser displays the page?

Style the heading

Now let's give those headings some style. How about changing the font a bit? Let's change both the type of font, and also the color of the heading fonts:

```
h1 {  
    font-family: sans-serif;  
    color: gray;  
}  
  
h2 {  
    font-family: sans-serif;  
    color: gray;  
}  
  
p {  
    color: maroon;  
}
```

Here's the rule to select `<h1>` elements and change the `font-family` to `sans-serif` and the `font-color` to `gray`. We'll talk a lot more about fonts later.

And here's another rule to do the exact same thing to the `<h2>` element.

How about a different font for the lounge headings?
Make them really stand out.
I'm seeing big, clean, gray...



Actually, because these rules are *exactly* the same, we can combine them, like this:

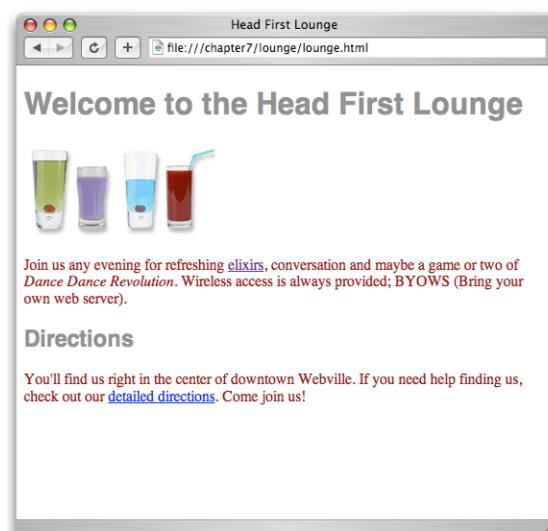
```
h1, h2 {  
    font-family: sans-serif;  
    color: gray;  
}  
  
p {  
    color: maroon;  
}
```

To write a rule for more than one element, just put commas between the selectors, like "h1, h2".

Test drive...

Add this new CSS to your "lounge.html" file and reload. You'll see that with one rule, you've selected both the `<h1>` and `<h2>` headings.

Both of the headings on the page are now styled with a sans-serif font and colored gray.



Let's put a line under the welcome message too

Let's touch up the welcome heading a bit more. How about a line under it? That should set the main heading apart visually and add a nice touch. Here's the property we'll use to do that:

```
border-bottom: 1px solid black;
```

This property controls how the border under an element looks.

We're going to style the bottom border so that it is a 1-pixel-thick, solid black line.

The trouble is, if we add this property and value to the combined `h1, h2` rule in our CSS, we'll end up with borders on both our headings:

```
h1, h2 {
    font-family: sans-serif;
    color: gray;
    border-bottom: 1px solid black;
}
```

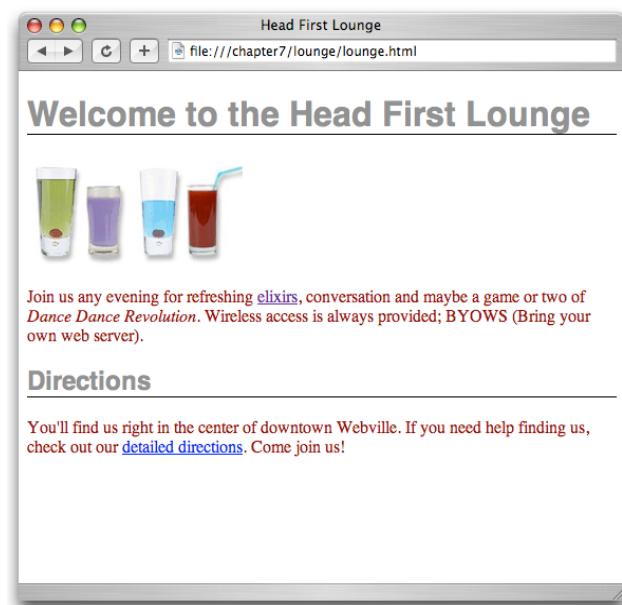
Here we're adding a property to change the bottom border for both the `<h1>` and `<h2>` elements.

```
p {
    color: maroon;
}
```

If we do this...

...we get bottom borders on both our headings. Not what we want.

So, how can we set the bottom border on just the `<h1>` element, without affecting the `<h2>` element? Do we have to split up the rules again? Turn the page to find out...



We have the technology: specifying a second rule, just for the `<h1>`

We don't have to split up the `h1`, `h2` rule, we just need to add another rule that is only for `h1` and add the border style to it.

```
h1, h2 {  
    font-family: sans-serif;  
    color: gray;  
}
```

The first rule stays the same. We're still going to use a combined rule for the `font-family` and `color` for both `<h1>` and `<h2>`.

```
h1 {  
    border-bottom: 1px solid black;  
}
```

But now we're adding a second rule that adds another property just to `<h1>`: the `border-bottom` property.

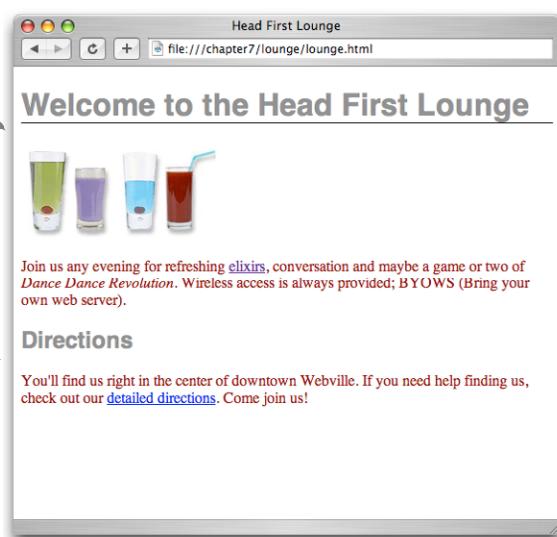
```
p {  
    color: maroon;  
}
```

Another test drive...

Change your CSS and reload the page. You'll see that the new rule added a black border to the bottom of the main heading, which gives us a nice underline on the heading and really makes it stand out.

Here's the
bottom border
in black.

And no border
here—just what
we wanted.



there are no Dumb Questions

Q: So how does that work when you have more than one rule for an element?

A: You can have as many rules as you want for an element. Each rule adds to the style information of the rule before it. In general, you try to group together all the common styles between elements, like we did with `<h1>` and `<h2>`, and then any style that is specific to an element, you write in another rule, like we did with the border-bottom style for the main heading.

Q: What's the advantage of that approach? Isn't it better to organize each element separately, so you know exactly what styles it has?

A: Not at all. If you combine common styles together, then if they change, you only have to change them in one rule. If you break them up, then there are many rules you have to change, which is error-prone.

Q: Why do we use a bottom border to underline text? Isn't there an underline style for text?

A: Good question. There is an underline style for text and we could use that instead. However, the two styles have slightly different effects on the page: if you use border-bottom, then the line will extend to the edge of the page. An underline is only shown under the text itself. The property to set text underline is called text-decoration and has a value of "underline" for underlined text. Give it a try and check out the differences.

So, how do selectors really work?

You've seen how to select an element to style it, like this:

```

h1 {
    color: gray;
}
  
```

We call this the selector.
The style is applied to the elements described by the selector—in this case, `<h1>` elements.

Or how to select more than one element, like this:

```

h1, h2 {
    color: gray;
}
  
```

Another selector. The style is applied to `<h1>` and `<h2>` elements.

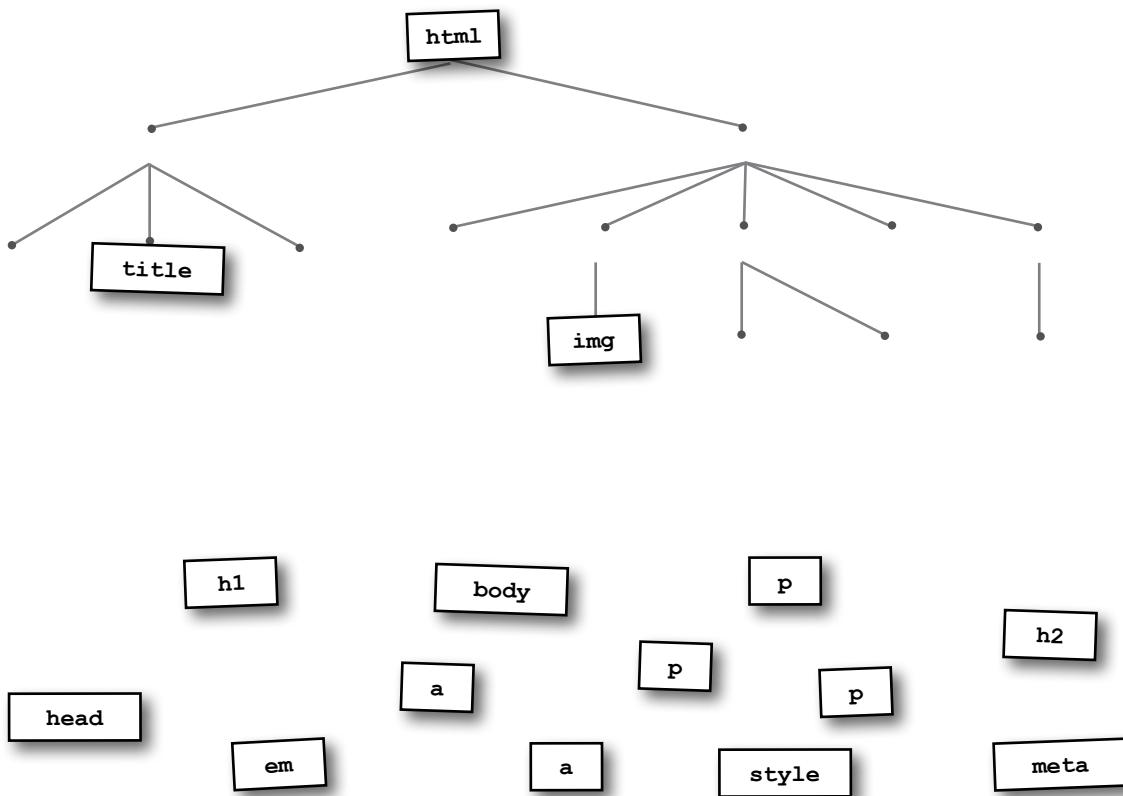
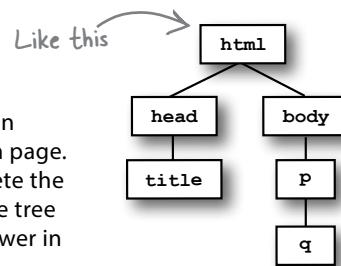
You're going to see that CSS allows you to specify all kinds of selectors that determine which elements your styles are applied to. Knowing how to use these selectors is the first step in mastering CSS, and to do that you need to understand the organization of the HTML that you're styling. After all, how can you select elements for styling if you don't have a good mental picture of what elements are in the HTML, and how they relate to one another?

So, let's get that picture of the lounge HTML in your head, and then we'll dive back into selectors.



Markup Magnets

Remember drawing the hierarchy diagram of HTML elements in Chapter 3? You're going to do that again for the Lounge's main page. Below, you'll find all the element magnets you need to complete the diagram. Using the Lounge's HTML (on the right), complete the tree below. We've done a couple for you already. You'll find the answer in the back of the chapter.



```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Head First Lounge</title>
    <style>
      h1, h2 {
        font-family: sans-serif;
        color: gray;
      }
      h1 {
        border-bottom: 1px solid black;
      }
      p {
        color: maroon;
      }
    </style>
  </head>
  <body>
    <h1>Welcome to the Head First Lounge</h1>
    <p>
      
    </p>
    <p>
      Join us any evening for refreshing
      <a href="beverages/elixir.html">elixirs</a>,
      conversation and maybe a game or two
      of <em>Dance Dance Revolution</em>.
      Wireless access is always provided;
      BYOWS (Bring your own web server).
    </p>
    <h2>Directions</h2>
    <p>
      You'll find us right in the center of downtown
      Webville. If you need help finding us, check out our
      <a href="about/directions.html">detailed directions</a>.
      Come join us!
    </p>
  </body>
</html>
```

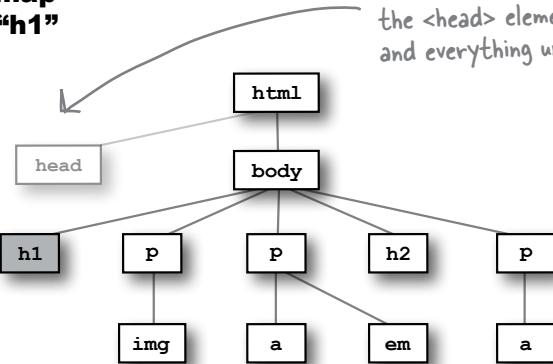
The Head First
Lounge HTML

Seeing selectors visually

Let's take some selectors and see how they map to the tree you just created. Here's how this "h1" selector maps to the graph:

```
h1 {  
    font-family: sans-serif;  
}
```

This selector matches any
<h1> elements in the page,
and there's only one.

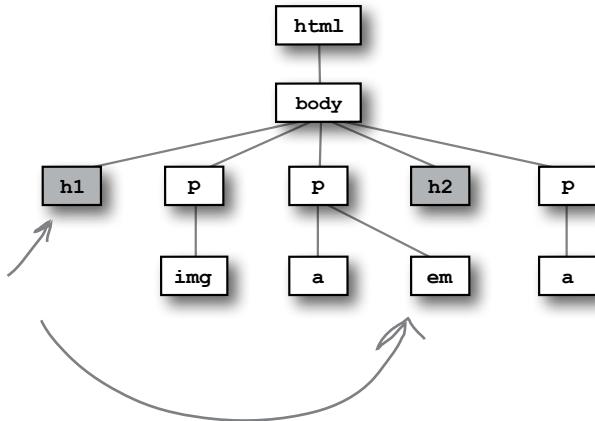


We can only style elements in the body,
so we're not showing the <head> element
and everything under it.

And here's how the "h1, h2" selector looks:

```
h1, h2 {  
    font-family: sans-serif;  
}
```

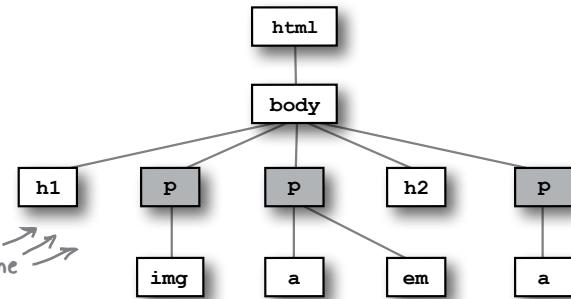
Now the selector
matches both <h1>
and <h2> elements.



If we use a "p" selector, here's how that looks:

```
p {  
    font-family: sans-serif;  
}
```

This selector matches all the
<p> elements in the tree.

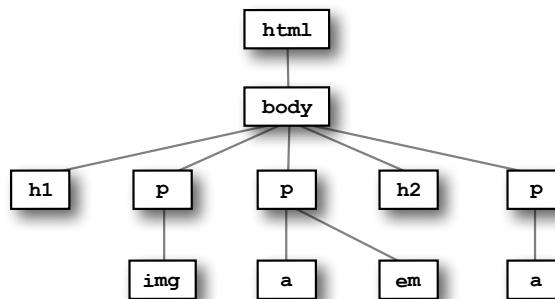




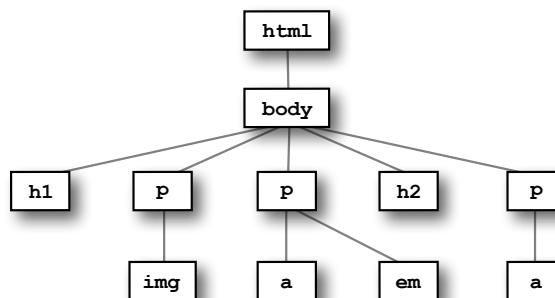
Sharpen your pencil

Color in the elements that are **selected** by these selectors:

```
p, h2 {  
    font-family: sans-serif;  
}
```



```
p, em {  
    font-family: sans-serif;  
}
```



Five-Minute Mystery



The Case of Brute Force Versus Style

When we last left RadWebDesign in Chapter 4, they had just blown the corporate demo and lost RobotsRUs's business.

CorrectWebDesign was put in charge of the entire RobotsRUs site and got to work getting everything nailed down before the site launch later in the month. But you'll also remember that

RadWebDesign decided to bone up on their HTML and CSS.

They decided to rework the RobotsRUs site on their own, using proper HTML and stylesheets, just to get some experience under their belt before they took on another consulting job.

As fate would have it, just before RobotsRUs's big site launch, it happened again: RobotsRUs called CorrectWebDesign with an urgent message. "We're changing our corporate look and we need all the colors, backgrounds, and fonts changed on our site." At this point, the site consisted of almost 100 pages, so CorrectWebDesign responded that it would take them a few days to rework the site. "We don't have a few days!" the CEO said. Desperate, the CEO decided to call in RadWebDesign for help. "You flubbed up the demo last month, but we really need your help. Can you help the CorrectWebDesign guys convert the site over to the new look and feel?" RadWebDesign said they could do better than that; in fact, they could deliver the entire site to them in less than an hour.

How did RadWebDesign go from disgrace to web page superheroes? What allowed them to change the look and feel of 100 pages faster than a speeding bullet?



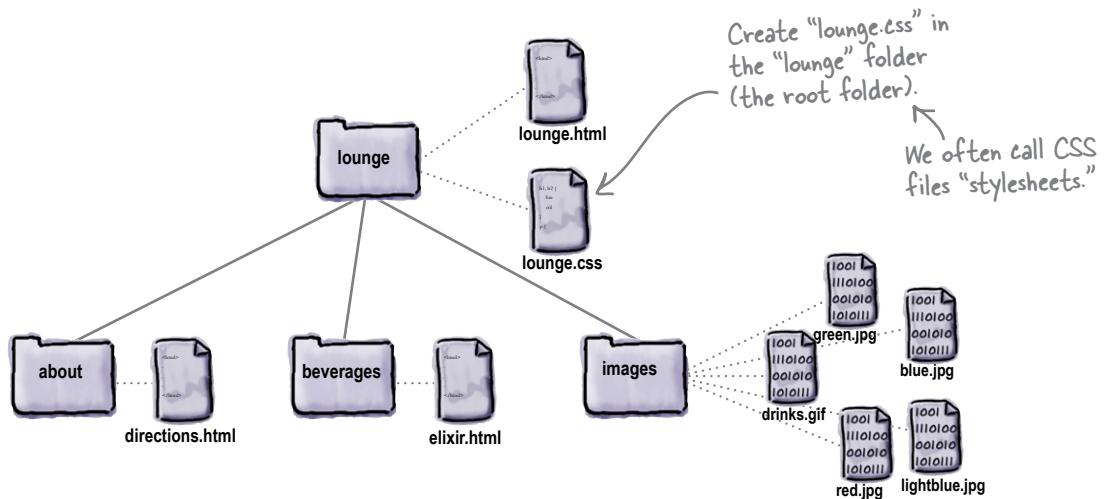
Getting the lounge style into the elixirs and directions pages

It's great that we've added all this style to "lounge.html", but what about "elixir.html" and "directions.html"? They need to have a look that is consistent with the main page. Easy enough...just copy the style element and all the rules into each file, right? *Not so fast*. If you did that, then whenever you needed to change the style of the site, you'd have to change *every single file*—not what you want. But luckily, there is a better way. Here's what you're going to do:

- ❶ Take the rules in "lounge.html" and place them in a file called "lounge.css".
- ❷ Create an **external link** to this file from your "lounge.html" file.
- ❸ Create the same external links in "elixir.html" and "directions.html".
- ❹ Give all three files a good test drive.

Creating the “lounge.css” file

You’re going to create a file called “lounge.css” to contain the style rules for all your Head First Lounge pages. To do that, create a new text file named “lounge.css” in your text editor.



Now type, or copy and paste from your “lounge.html” file, the CSS rules into the “lounge.css” file. Delete the rules from your “lounge.html” file while you’re at it.

Note that you should not copy the `<style>` and `</style>` tags because the “lounge.css” file contains only CSS, not HTML.

```
h1, h2 {  
    font-family: sans-serif;  
    color: gray;  
}  
  
h1 {  
    border-bottom: 1px solid black;  
}  
  
p {  
    color: maroon;  
}
```

Your “lounge.css” file
should look like this.
Remember, no `<style>` tags!

Linking from “lounge.html” to the external stylesheet

Now we need a way to tell the browser that it should style this page with the styles in the external stylesheet. We can do that with the HTML `<link>` element. Here’s how you use the `<link>` element in your HTML:

```

<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Head First Lounge</title>
    <link type="text/css" rel="stylesheet" href="lounge.css">
    <style>
    </style>
  </head>
  <body>
    <h1>Welcome to the Head First Lounge</h1>
    <p>
      
    </p>
    .
    .
  </p>
</body>
</html>

```



HTML Up Close

Let’s take a closer look at the `<link>` element since you haven’t seen it before:

Use the `link` element to “link in” external information.

The type of this information is “`text/css`”—in other words, a CSS stylesheet. As of HTML5, you don’t need this anymore (it’s optional), but you may see it on older pages.

And the stylesheet is located at this `href` (in this case, we’re using a relative link, but it could be a full-blown URL).

```
<link type="text/css" rel="stylesheet" href="lounge.css">
```

The `rel` attribute specifies the relationship between the HTML file and the thing you’re linking to. We’re linking to a stylesheet, so we use the value “`stylesheet`”.

`<link>` is a void element. It has no closing tag.

Linking from “elixir.html” and “directions.html” to the external stylesheet

Now you’re going to link the “elixir.html” and “directions.html” files just as you did with “lounge.html”. The only thing you need to remember is that “elixir.html” is in the “beverages” folder, and “directions.html” is in the “about” folder, so they both need to use the relative path “..../lounge.css”.

So, all you need to do is add the following <link> element to both files:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Head First Lounge Elixirs</title>
    <link type="text/css" rel="stylesheet" href="../lounge.css">
  </head>
  <body>
    .
    .
    .
  </body>
</html>
```



This is “elixir.html”. Just add the <link> line.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Head First Lounge Directions</title>
    <link type="text/css" rel="stylesheet" href="../lounge.css">
  </head>
  <body>
    .
    .
    .
  </body>
</html>
```

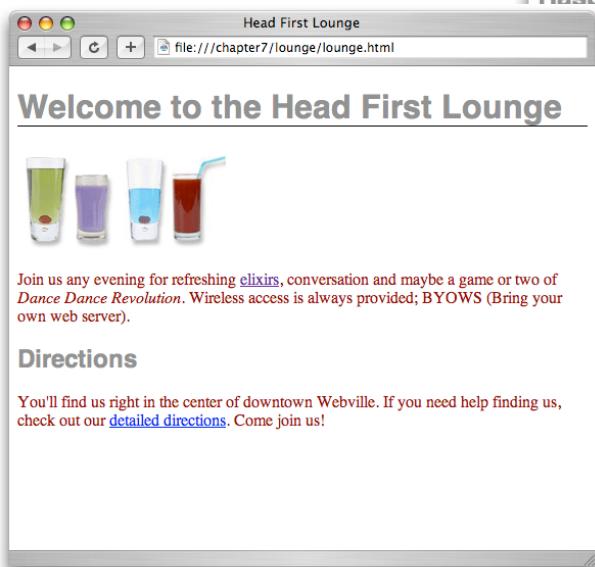


Same for “directions.html”. Add the <link> line here.

Test driving the entire lounge...

Save each of these files and then open “lounge.html” with the browser. You should see no changes in its style, even though the styles are now coming from an external file. Now click on the “elixirs” and “detailed directions” links.

Wow! We have a whole new style for the Elixirs and Directions pages with only a *one-line change* to the HTML in each file! Now you can really see the power of CSS.



The screenshot shows a web browser window titled "Head First Lounge Elixirs". The URL in the address bar is "file:///chapter7/lounge/beverages/elixir.html". The page content features a heading "Our Elixirs" and two sections: "Green Tea Cooler" and "Raspberry Ice Concentration". Each section includes a small image of the elixir and a brief description. The "Green Tea Cooler" description reads: "Chock full of vitamins and minerals, this elixir combines the healthful benefits of green tea with a twist of chamomile blossoms and ginger root."

The screenshot shows a web browser window titled "Head First Lounge Directions". The URL in the address bar is "file:///chapter7/lounge/about/directions.html". The page content includes a heading "Directions" and a series of step-by-step driving instructions:

- Take the 305 S exit to Webville - go 0.4 mi
- Continue on 305 - go 12 mi
- Turn right at Structure Ave N - go 0.6 mi
- Turn right and head toward Structure Ave N - go 0.0 mi
- Turn right at Structure Ave N - go 0.7 mi
- Continue on Stucture Ave S - go 0.2 mi
- Turn right at SW Presentation Way - go 0.0 mi

Five-Minute Mystery Solved



The Case of Brute Force Versus Style

So, how did RadWebDesign become web page superheroes? Or maybe we should first ask how the “do no wrong” CorrectWebDesign firm flubbed things up this time? The root of the problem was that CorrectWebDesign was creating the RobotsRUs pages using circa-1998 techniques. They were putting their style rules right in with their HTML (copying and pasting them each time), and, even worse, they were using a lot of old HTML elements like `` and `<center>` that have now been deprecated (eliminated from HTML). So, when the call came to change the look and feel, that meant going into every web page and making changes to the CSS. Worse, it meant going through the HTML to change elements as well.

Compare that with what RadWebDesign did: they used HTML5, so they had no old presentation HTML in their pages, and they used an external stylesheet. The result? To change the style of the entire site, all they had to do was go into their external stylesheet and make a few changes to the CSS, which they easily did in minutes, not days. They even had time to try out multiple designs and have three different versions of the CSS ready for review before the site launch. Amazed, the RobotsRUs CEO not only promised RadWebDesign more business, but he also promised them the first robot that comes off the assembly line.

Sharpen your pencil

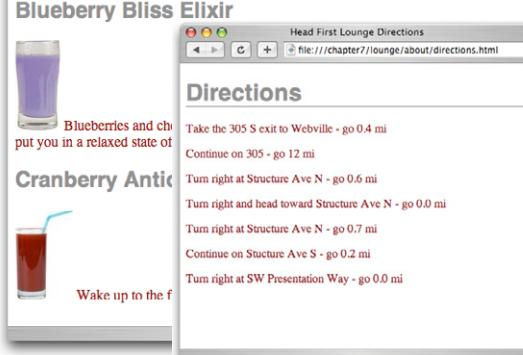
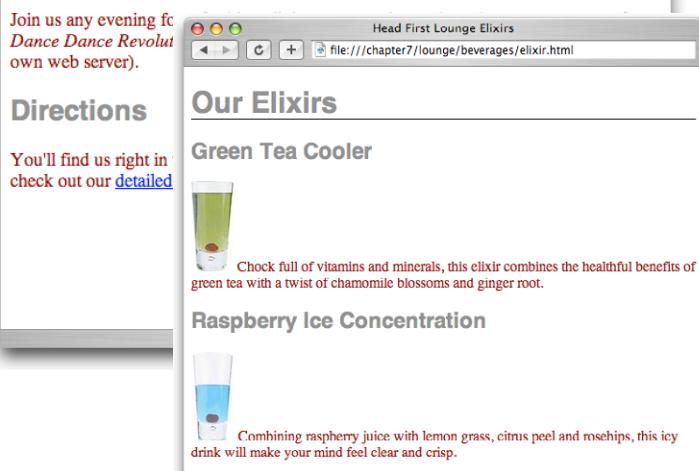
Now that you've got one external style file (or "stylesheet"), use it to change all the paragraph fonts to "sans-serif" to match the headings. Remember, the property to change the font style is "font-family", and the value for sans-serif font is "sans-serif". You'll find the answer on the next page.

The headings use sans-serif fonts, which don't have serifs and have a very clean look.

The paragraphs still use the default serif fonts, which have serifs, and are often considered more difficult to read on a computer screen.

any

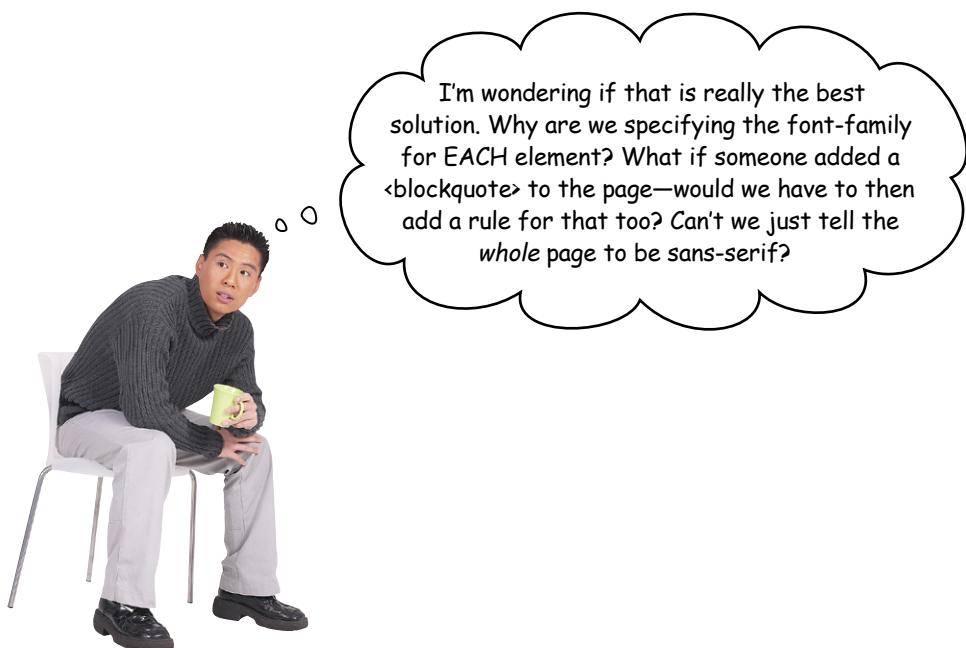
Serifs



Sharpen your pencil Solution

Now that you've got one external style file (or "stylesheet"), use it to change all the paragraph fonts to "sans-serif" to match the headings. Remember, the property to change the font style is "font-family", and the value for sans-serif font is "sans-serif". Here's our solution.

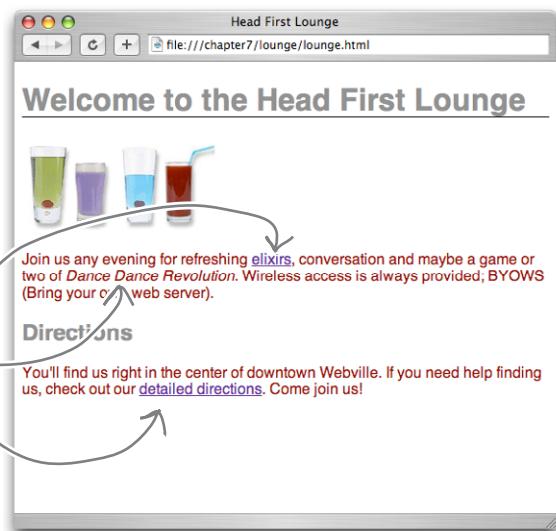
```
h1, h2 {  
    font-family: sans-serif;  
    color: gray;  
}  
  
h1 {  
    border-bottom: 1px solid black;  
}  
  
p {  
    font-family: sans-serif; ← Just add a font-family property  
    color: maroon;  
}
```



It's time to talk about your inheritance...

Did you notice when you added the `font-family` property to your `p` selector that it also affected the font family of the elements inside the `<p>` element? Let's take a closer look:

When you added the `font-family` property to your CSS `p` selector, it changed the font family of your `<p>` elements. But it also changed the font family of the two links and the emphasized text.

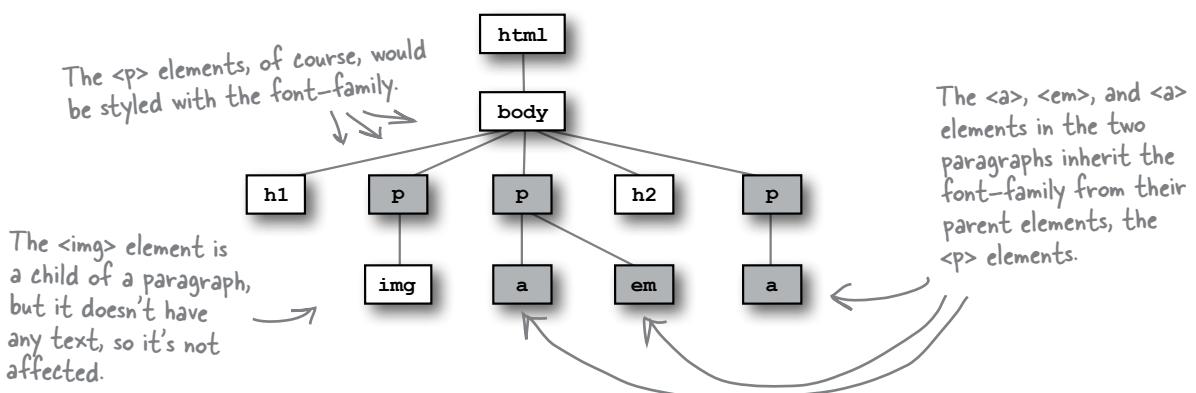


The elements inside the `<p>` element inherit the font-family style from `<p>`

Just like you can inherit your blue eyes or brown hair from your parents, elements can inherit styles from their parents. In this case, the `<a>` and `` elements inherited the `font-family` style from the `<p>` element, which is their parent element. It makes sense that changing your paragraph style would change the style of the elements in the paragraph, doesn't it? After all, if it didn't, you'd have to go in and add CSS rules for every inline element in every paragraph in your whole site...which would definitely be so NOT fun.

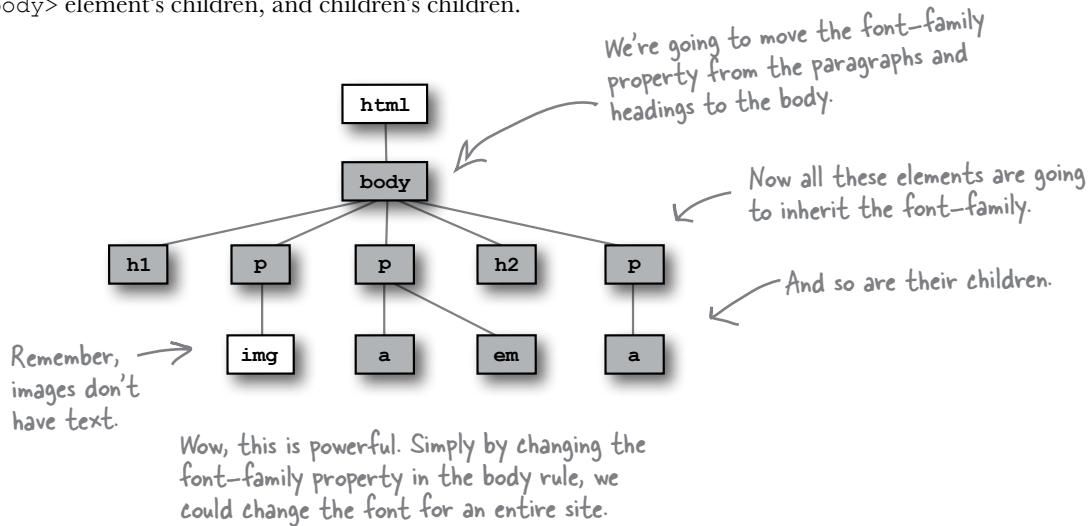
Let's take a look at the HTML tree to see how inheritance works:

If we set the `font-family` of all the `<p>` elements, here are all the elements that would be affected.



What if we move the font up the family tree?

If most elements inherit the `font-family` property, what if we move it up to the `<body>` element? That should have the effect of changing the font for all the `<body>` element's children, and children's children.



What are you waiting for...give it a try

Open your “lounge.css” file and add a new rule that selects the `<body>` element. Then remove the `font-family` properties from the headings and paragraph rules, because you’re not going to need them anymore.

```
body {  
    font-family: sans-serif;  
}  
  
h1, h2 {  
    font-family: sans-serif;  
    color: gray;  
}  
  
h1 {  
    border-bottom: 1px solid black;  
}  
  
p {  
    font-family: sans-serif;  
    color: maroon;  
}
```

Here's what you're going to do.

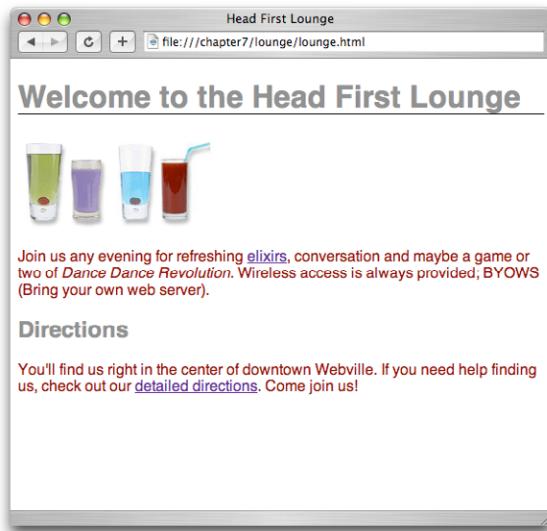
First, add a new rule that selects the `<body>` element. Then add the `font-family` property with a value of `sans-serif`.

Then, take the `font-family` property out of the `h1`, `h2` rule, as well as the `p` rule.

Test drive your new CSS

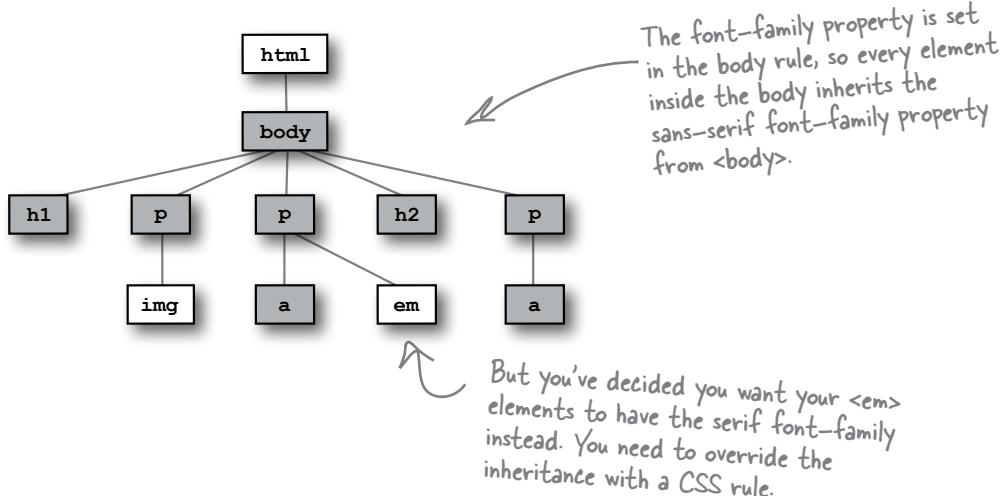
As usual, go ahead and make these changes in the “lounge.css” stylesheet, save, and reload the “lounge.html” page. You shouldn’t expect any changes, because the style is the same. It’s just coming from a different rule. But you should feel better about your CSS, because now you can add new elements to your pages and they’ll automatically inherit the sans-serif font.

Surprise, surprise. This doesn’t look any different at all, but that is exactly what we were expecting, isn’t it? All you’ve done is move the sans-serif font up into the body rule and let all the other elements inherit that.



Overriding inheritance

By moving the `font-family` property up into the `body`, you've set that font style for the entire page. But what if you don't want the sans-serif font on every element? For instance, you could decide that you want `` elements to use the serif font instead.



Well, then you can override the inheritance by supplying a specific rule just for ``. Here's how you add a rule for `` to override the `font-family` specified in the `body`:

```
body {  
    font-family: sans-serif;  
}  
  
h1, h2 {  
    color: gray;  
}  
  
h1 {  
    border-bottom: 1px solid black;  
}  
  
p {  
    color: maroon;  
}  
  
em {  
    font-family: serif;  
}
```

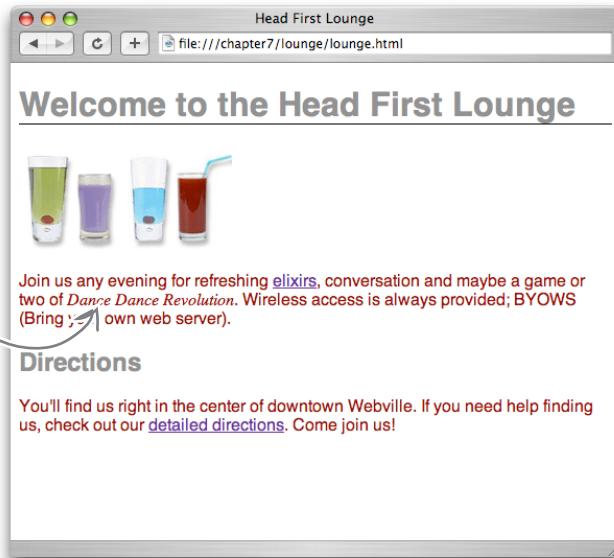
To override the `font-family` property inherited from `body`, add a new rule selecting `em` with the `font-family` property value set to `serif`.

Test drive

Add a rule for the `` element to your CSS with a `font-family` property value of `serif`, and reload your “lounge.html” page:

Notice that the “Dance Dance Revolution” text, which is the text in the `` element, is now a serif font.

As a general rule, it’s not a good idea to change fonts in the middle of a paragraph like this, so go ahead and change your CSS back to the way it was (without the em rule) when you’re done testing.



there are no Dumb Questions

Q: How does the browser know which rule to apply to `` when I’m overriding the inherited value?

A: With CSS, the most specific rule is always used. So, if you have a rule for `<body>`, and a more specific rule for `` elements, it is going to use the more specific rule. We’ll talk more later about how you know which rules are most specific.

Q: How do I know which CSS properties are inherited and which are not?

A: This is where a good reference really comes in handy, like O’Reilly’s *CSS Pocket Reference*. In general, all of the styles that affect the way your text looks, such as font color (the `color` property), the `font-family`, as you’ve just seen, and other font-related properties such as `font-size`, `font-weight` (for bold text), and `font-style` (for italics) are

inherited. Other properties, such as border, are not inherited, which makes sense, right? Just because you want a border on your `<body>` element doesn’t mean you want it on *all* your elements. A lot of the time, you can follow your common sense (or just try it and see), and you’ll get the hang of it as you become more familiar with the various properties and what they do.

Q: Can I always override a property that is being inherited when I don’t want it?

A: Yes. You can always use a more specific selector to override a property from a parent.

Q: This stuff gets complicated. Is there any way I can add comments to remind myself what the rules do?

A: Yes. To write a comment in your CSS, just enclose it between `/*` and `*/`. For instance:

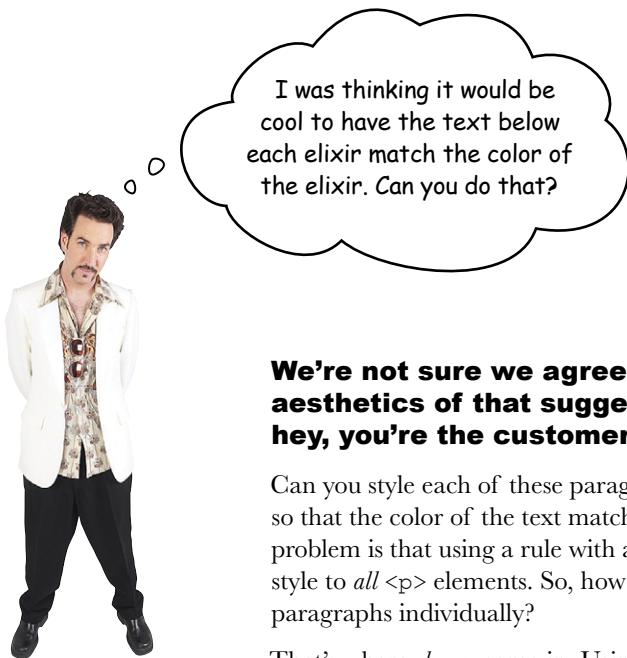
```
/* this rule selects all
paragraphs and colors them
blue */
```

Notice that a comment can span multiple lines. You can also put comments around CSS and browsers will ignore it, like this:

```
/* this rule will have no
effect because it's in a
comment */
```

```
p { color: blue; } */
```

Make sure you close your comments correctly; otherwise, your CSS won’t work!



We're not sure we agree with the aesthetics of that suggestion, but hey, you're the customer.

Can you style each of these paragraphs separately so that the color of the text matches the drink? The problem is that using a rule with a `p` selector applies the style to *all* `<p>` elements. So, how can you select these paragraphs individually?

That's where *classes* come in. Using both HTML and CSS, we can define a class of elements, and then apply styles to any element that belongs to that class. So, what exactly is a class? Think of it like a club—someone starts a “greentea” club, and by joining you agree to all the rights and responsibilities of the club, like adhering to their style standards. Anyway, let's just create the class and you'll see how it works.

Creating a class takes two steps: first, we add the element to the class by adding a `class` attribute to the element in the HTML; second, we select that class in the CSS. Let's step through that...

Green text →

Blue text →

Purple text →

Red text...oh, we
don't need to
change this one. →

Head First Lounge Elixirs
file:///chapter7/lounge/beverages/elixir.html

Our Elixirs

Green Tea Cooler
Chock full of vitamins and minerals, this elixir combines the healthful benefits of green tea with a twist of chamomile blossoms and ginger root.

Raspberry Ice Concentration
Combining raspberry juice with lemon grass, citrus peel and rosehips, this icy drink will make your mind feel clear and crisp.

Blueberry Bliss Elixir
Blueberries and cherry essence mixed into a base of elderflower herb tea will put you in a relaxed state of bliss in no time.

Cranberry Antioxidant Blast
Wake up to the flavors of cranberry and hibiscus in this vitamin C rich elixir.

Adding an element to the greentea class

Open up the “elixir.html” file and locate the “Green Tea Cooler” paragraph. This is the text we want to change to green. All you’re going to do is add the `<p>` element to a class called `greentea`. Here’s how you do that:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Head First Lounge Elixirs</title>
    <link type="text/css" rel="stylesheet" href="../lounge.css">
  </head>
  <body>
    <h1>Our Elixirs</h1>
    <h2>Green Tea Cooler</h2>
    <p class="greentea">
      
      Chock full of vitamins and minerals, this elixir
      combines the healthful benefits of green tea with
      a twist of chamomile blossoms and ginger root.
    </p>
    <h2>Raspberry Ice Concentration</h2>
    <p>
      
      Combining raspberry juice with lemon grass,
      citrus peel and rosehips, this icy drink
      will make your mind feel clear and crisp.
    </p>
    <h2>Blueberry Bliss Elixir</h2>
    <p>
      
      Blueberries and cherry essence mixed into a base
      of elderflower herb tea will put you in a relaxed
      state of bliss in no time.
    </p>
    <h2>Cranberry Antioxidant Blast</h2>
    <p>
      
      Wake up to the flavors of cranberry and hibiscus
      in this vitamin C rich elixir.
    </p>
  </body>
</html>

```

To add an element to a class, just add the attribute “`class`” along with the name of the class, like “`greentea`”.

Now that the green tea paragraph belongs to the `greentea` class, you just need to provide some rules to style that class of elements.

Creating a class selector

To create a class in CSS and select an element in that class, you write a *class selector*, like this:

The diagram shows a CSS rule with handwritten annotations:

- An arrow points from the text "Select the element in the class first—in this case, p." to the selector `p.greentea`.
- A callout bubble above the selector says "Then use a ‘.’ to specify a class." with an arrow pointing to the dot.
- A callout bubble next to the class name says "Last is the class name." with an arrow pointing to the word "greentea".
- A bracket under the selector `p.greentea` is labeled "The selector p.greentea selects all paragraphs in the greentea class."
- A callout bubble to the right of the rule says "And here's the rule...make any text in a paragraph in the greentea class the color green."

```
p.greentea {  
    color: green;  
}
```

So now you have a way of selecting `<p>` elements that belong to a certain class to style them. All you need to do is add the `class` attribute to any `<p>` elements you want to be green, and this rule will be applied. Give it a try: open your “lounge.css” file and add the `p.greentea` class selector to it.

```
body {  
    font-family: sans-serif;  
}  
  
h1, h2 {  
    color: gray;  
}  
  
h1 {  
    border-bottom: 1px solid black;  
}  
  
p {  
    color: maroon;  
}  
  
p.greentea {  
    color: green;  
}
```

A greentea test drive

Save and then reload to give your new class a test drive.

Here's the new greentea class applied to the paragraph. Now the font is green and matches the Green Tea Cooler. Maybe this styling wasn't such a bad idea after all.



Sharpen your pencil

Your turn: add two classes, "raspberry" and "blueberry", to the correct paragraphs in "elixir.html", and then write the styles to color the text blue and purple, respectively. The property value for raspberry is "blue" and for blueberry is "purple". Put these at the bottom of your CSS file, under the greentea rule: raspberry first, and then blueberry.

Yeah, we know you're probably thinking, how can a raspberry be blue? Well, if Raspberry Kool-Aid is blue, that's good enough for us. And seriously, when you blend up a bunch of blueberries, they really are more purple than blue. Work with us here.

Taking classes further...

You've already written one rule that uses the `greentea` class to change any paragraph in the class to the color "green":

```
p.greentea {  
    color: green;  
}
```

But what if you wanted to do the same to all `<blockquote>`s?

Then you could do this:

```
blockquote.greentea, p.greentea {  
    color: green;  
}
```

Just add another selector to handle `<blockquote>`s that are in the `greentea` class. Now this rule will apply to `<p>` and `<blockquote>` elements in the `greentea` class.

And in your HTML you'd write:

```
<blockquote class="greentea">
```

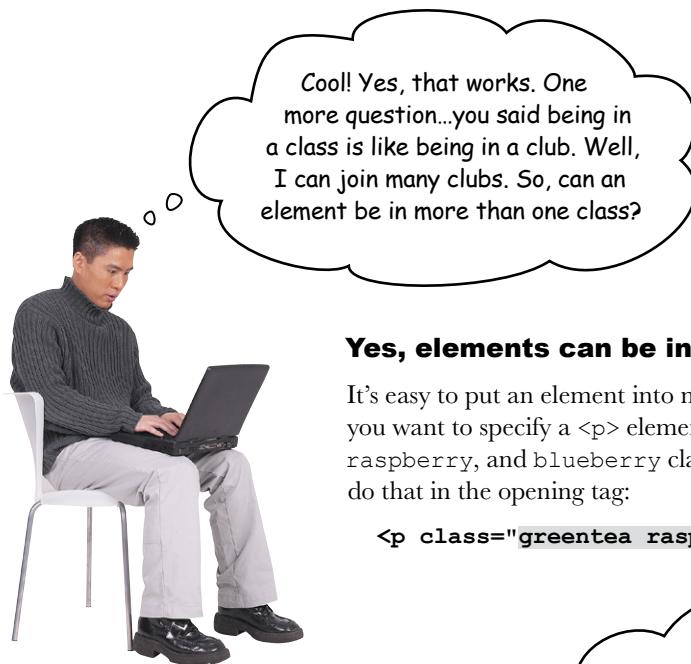


So what if I want to add `<h1>`, `<h2>`, `<h3>`, `<p>`, and `<blockquote>` to the `greentea` class? Do I have to write one huge selector?

No, there's a better way. If you want all elements that are in the `greentea` class to have a style, then you can just write your rule like this:

```
.greentea {  
    color: green;  
}
```

If you leave out all the element names, and just use a period followed by a class name, then the rule will apply to all members of the class.

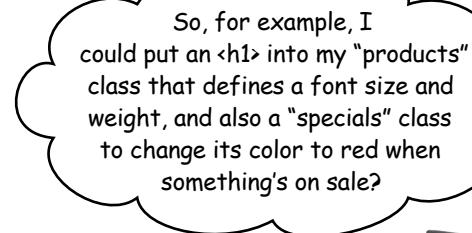


Yes, elements can be in more than one class.

It's easy to put an element into more than one class. Say you want to specify a `<p>` element that is in the `greentea`, `raspberry`, and `blueberry` classes. Here's how you would do that in the opening tag:

```
<p class="greentea raspberry blueberry">
```

Place each class name into the value of the `class` attribute, with a space in between each. The ordering doesn't matter.



Exactly. Use multiple classes when you want an element to have styles you've defined in different classes. In this case, all your `<h1>` elements associated with products have a certain style, but not all your products are on sale at the same time. By putting your "specials" color in a separate class, you can simply add only those elements associated with products on sale to the "specials" class to add the red color you want.



Now you may be wondering what happens when an element belongs to multiple classes, all of which define the *same* property—like our `<p>` element up there. How do you know which style gets applied? You know each of these classes has a definition for the `color` property. So, will the paragraph be green, blue (`raspberry`), or purple?

We're going to talk about this in great detail after you've learned a bit more CSS, but on the next page you'll find a quick guide to hold you over.

The world's smallest and fastest guide to how styles are applied

Elements and document trees and style rules and classes...it can get downright confusing. How does all this stuff come together so that you know which styles are being applied to which elements? As we said, *to fully answer that* you're going to have to know a little more about CSS, and you'll be learning that in the next few chapters. But before you get there, let's just walk through some common-sense rules of thumb about how styles are applied.

First, do any selectors select your element?

Let's say you want to know the `font-family` property value for an element. The first thing to check is: is there a selector in your CSS file that selects your element? If there is, and it has a `font-family` property and value, then that's the value for your element.

What about inheritance?

If there are no selectors that match your element, then you rely on inheritance. So, look at the element's parents, and parents' parents, and so on, until you find the property defined. When and if you find it, that's the value.

Struck out again? Then use the default

If your element doesn't inherit the value from any of its ancestors, then you use the default value defined by the browser. In reality, this is a little more complicated than we're describing here, but we'll get to some of those details later in the book.

What if multiple selectors select an element?

Ah, this is the case we have with the paragraph that belongs to all three classes:

```
<p class="greentea raspberry blueberry">
```

There are multiple selectors that match this element and define the same `color` property. That's what we call a *conflict*. Which rule breaks the tie? Well, if one rule is more *specific* than the others, then it wins. But what does "more specific" mean? We'll come back in a later chapter and see *exactly* how to determine how specific a selector is, but for now, let's look at some rules and get a feel for it:

```
p { color: black; }  
.greentea { color: green; }  
p.greentea { color: green; }  
p.raspberry { color: blue; }  
p.blueberry { color: purple; }
```

Here's a rule that selects any old paragraph element.

This rule selects members of the greentea class. That's a little more specific.

And this rule selects only paragraphs that are in the greentea class, so that's even more specific.

These rules also select only paragraphs in a particular class. So they are about the same in specificity as the p.greentea rule.

And if we still don't have a clear winner?

So, if you had an element that belonged only to the `greentea` class, there would be an obvious winner: the `p.greentea` selector is the most specific, so the text would be green. But you have an element that belongs to *all three* classes: `greentea`, `raspberry`, and `blueberry`. So, `p.greentea`, `p.raspberry`, and `p.blueberry` all select the element, and are of equal specificity. What do you do now? You choose the one that is listed *last* in the CSS file. If you can't resolve a conflict because two selectors are equally specific, you use the ordering of the rules in your stylesheet file; that is, you use the rule listed last in the CSS file (nearest the bottom). And in this case, that would be the `p.blueberry` rule.



Exercise

In your “elixir.html” file, change the `greentea` paragraph to include all the classes, like this:

```
<p class="greentea raspberry blueberry">
```

Save and reload. What color is the Green Tea Cooler paragraph now? _____

Next, reorder the classes in your HTML:

```
<p class="raspberry blueberry greentea">
```

Save and reload. What color is the Green Tea Cooler paragraph now? _____

Next, open your CSS file and move the `p.greentea` rule to the bottom of the file.

Save and reload. What color is the Green Tea Cooler paragraph now? _____

Finally, move the `p.raspberry` rule to the bottom of the file.

Save and reload. What color is the Green Tea Cooler paragraph now? _____

After you've finished, rewrite the `green tea` element to look like it did originally:

```
<p class="greentea">
```

Save and reload. What color is the Green Tea Cooler paragraph now? _____

Fireside Chats



Tonight's talk: **CSS & HTML compare languages**

CSS:

Did you see that? I'm like Houdini! I broke right out of your `<style>` element and into my own file. And you said in Chapter 1 that I'd never escape.

Have to link me in? Come on; you know your pages wouldn't cut it without my styling.

If you were paying attention in this chapter, you would have seen I'm downright powerful in what I can do.

Well now, that's a little better. I like the new attitude.

HTML:

Don't get all excited; I still have to link you in for you to be at all useful.

Here we go again...while me and all my elements are trying to keep things structured, you're talking about hair highlights and nail color.

Okay, okay, I admit it; using CSS sure makes my job easier. All those old deprecated styling elements were a pain in my side. I do like the fact that my elements can be styled without inserting a bunch of stuff in the HTML, other than maybe an occasional class attribute.

But I still haven't forgotten how you mocked my syntax...`<remember>`?

CSS:

You have to admit HTML is kinda clunky, but that's what you get when you're related to an early '90s technology.

Are you kidding? I'm very expressive. I can select just the elements I want, and then describe exactly how I want them styled. And you've only just begun to see all the cool styling I can do.

Yup; just wait and see. I can style fonts and text in all kinds of interesting ways. I can even control how each element manages the space around it on the page.

Bwahahaha. And you thought you had me controlled between your `<style>` tags. You're going to see I can make your elements sit, bark, and roll over if I want to.

HTML:

I call it standing the test of time. And you think CSS is elegant? I mean, you're just a bunch of rules. How's that a language?

Oh yeah?

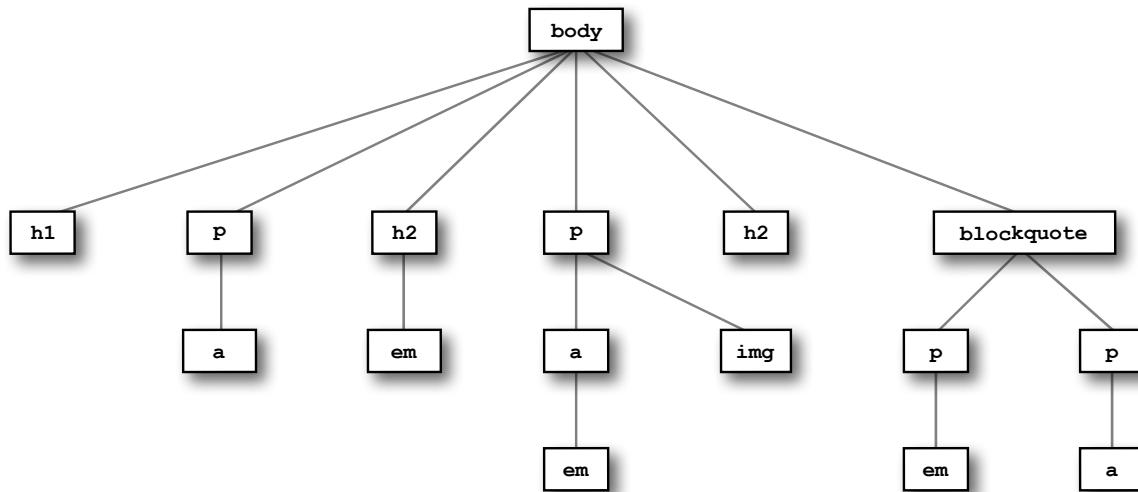
Hmm...sounds as if you have a little too much power; I'm not sure I like the sound of that. After all, my elements want to have some control over their own lives.

Whoa now! Security...security?!



Who gets the inheritance?

Sniff, sniff; the <body> element has gone to that great browser in the sky. But he left behind a lot of descendants and a big inheritance of color “green”. Below, you’ll find his family tree. Mark all the descendants that inherit the <body> element’s color green. Don’t forget to look at the CSS below first.



```
body {  
    color: green;  
}  
  
p {  
    color: black;  
}
```

Here's the CSS. Use this to determine which of the above elements hit the jackpot and get the green (color).

If you have errors in your CSS, usually what happens is all the rules below the error are ignored. So, get in the habit of looking for errors now, by doing this exercise.

BE the Browser

Below, you'll find the CSS file "style.css", with some errors in it. Your job is to play like you're the browser and locate all the errors. After you've done the exercise, look at the end of the chapter to see if you caught all the errors.



The file "style.css"

```
<style>

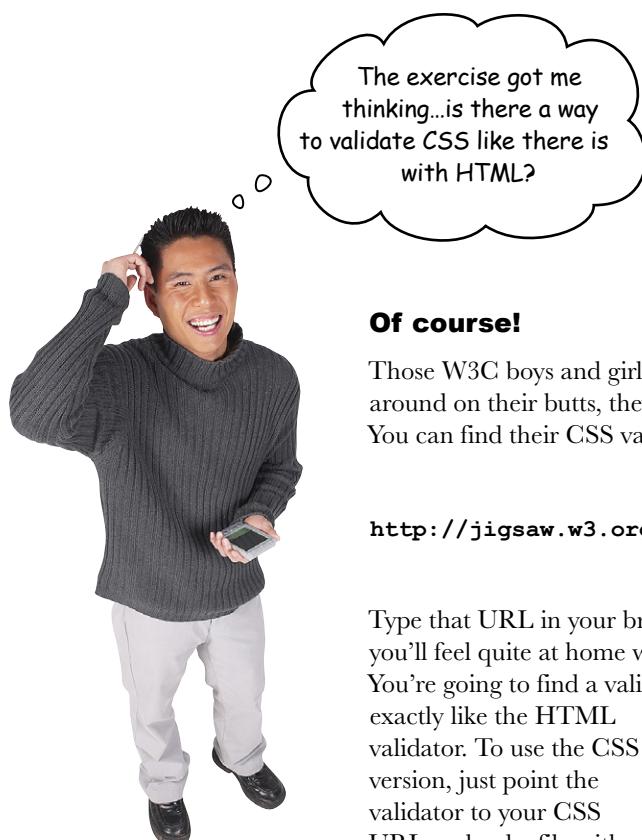
body {
    background-color: white
}

h1, {
    gray;
    font-family: sans-serif;
}

h2, p {
    color:
}

<em> {
    font-style: italic;
}

</style>
```



Of course!

Those W3C boys and girls aren't just sitting around on their butts, they've been working hard. You can find their CSS validator at:

<http://jigsaw.w3.org/css-validator/>

Type that URL in your browser, and we think you'll feel quite at home when you get there. You're going to find a validator that works almost exactly like the HTML validator. To use the CSS version, just point the validator to your CSS URL, upload a file with your CSS in it (first tab), or just paste it into the form (second tab), and submit.

You shouldn't encounter any big surprises, like needing doctypes or character encodings with CSS. Go ahead, give it a try (like we're not going to make you do it on the next page, anyway).

The W3C CSS Validation Service

Deutsch English Español Français 한국어 Italiano Nederlands 日本語 Polski Português Русский Svenska Български Українська Česština Romanian Magyar Ελληνικά 简体中文

CSS Validation Service

Check Cascading Style Sheets (CSS) and (X)HTML documents with style sheets

By URI By file upload By direct input

Validate by URI

Enter the URI of a document (HTML with CSS or CSS only) you would like validated:

Address:

More Options

Check

I ❤ VALIDATOR

The W3C validators rely on community support for hosting and development. 3029

Donate and help us build better tools for a better web. Flat

Note: If you want to validate your CSS style sheet embedded in an (X)HTML document, you should first check that the (X)HTML you use is valid.

About Documentation Download Feedback Credits

QUALITY ASSURANCE CSS

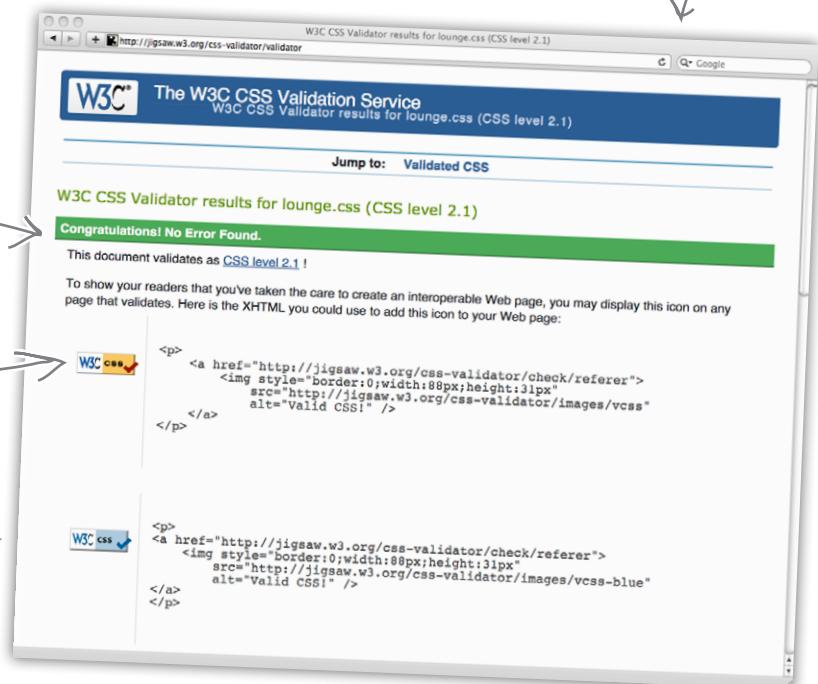
COPYRIGHT © 1994-2009 W3C® (MIT, ERCIM, KEIO). ALL RIGHTS RESERVED. W3C LIABILITY, TRADEMARK, DOCUMENT USE AND SOFTWARE LICENSING RULES APPLY. YOUR INTERACTIONS WITH THIS SITE ARE IN ACCORDANCE WITH OUR PUBLIC AND MEMBER PRIVACY STATEMENTS.

Making sure the lounge CSS validates

Before you wrap up this chapter, wouldn't you feel a lot better if all that Head First Lounge CSS validated? Sure, you would. Use whichever method you want to get your CSS to the W3C. If you have your CSS on a server, type your URL into the form; otherwise, either upload your CSS file or just copy and paste the CSS into the form. (If you upload, make sure you're directing the form to your CSS file, not your HTML file.) Once you've done that, click on Check.

If your CSS didn't validate, check it with the CSS a few pages back and find any small mistakes you've made, then resubmit.

Vay! Our CSS validates as CSS 2.1 (the validator hasn't upgraded to CSS 3 yet, but if it has by the time you read this, it should still validate).
 Here are some icons you can put on your web page if you want to show off that your CSS validates. (You can get similar icons for validated HTML, too.)



there are no
Dumb Questions

Q: Do I need to worry if I get warnings? Or do what they say?

A: It's good to look them over, but you'll find some are more in the category of suggestions than "must do's." The validator can err on the side of being a little anal, so just keep that in mind.

Just like when you validate HTML correctly, you get the "green badge of success" when you pass validation for your CSS. Green is good!

Property Soup

Use color to set the font color of text elements.

color

This property controls the weight of text. Use it to make text bold.

font-weight

This is how you tell an element how to position its left side.

This property sets the space between lines in a text element.

line-height

top
Controls the position of the top of the element.

letter-spacing

text-align

Use this property to align your text to the left, center, or right.

This lets you set the spacing between letters. Like this.

background-color

This property controls the background color of an element.

border

This property puts a border around an element. You can have a solid border, a ridged border, a dotted border...

padding

If you need space between the edge of an element and its content, use padding.

Makes text bigger or smaller.

font-size

Use this property for italic or oblique text.

font-style

This property lets you change how list items look in a list.

list-style

Use this property to put an image behind an element.

background-image

CSS has a **lot of style properties**. You'll see quite a few of these in the rest of this book, but have a quick look now to get an idea of all the aspects of style you can control with CSS.



BULLET POINTS

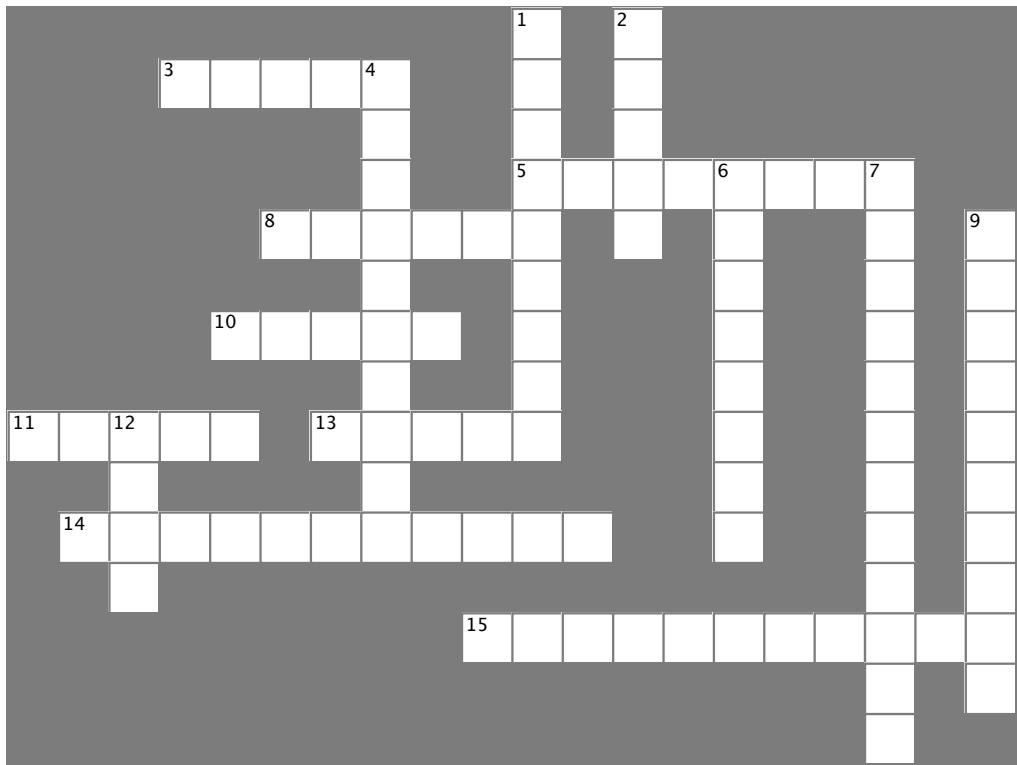


- CSS contains simple statements, called rules.
- Each rule provides the style for a selection of HTML elements.
- A typical rule consists of a selector along with one or more properties and values.
- The selector specifies which elements the rule applies to.
- Each property declaration ends with a semicolon.
- All properties and values in a rule go between {} braces.
- You can select any element using its name as the selector.
- By separating element names with commas, you can select multiple elements at once.
- One of the easiest ways to include a style in HTML is the `<style>` tag.
- For HTML and for sites of any complexity, you should link to an external stylesheet.
- The `<link>` element is used to include an external stylesheet.
- Many properties are inherited. For instance, if a property that is inherited is set for the `<body>` element, all the `<body>`'s child elements will inherit it.
- You can always override properties that are inherited by creating a more specific rule for the element you'd like to change.
- Use the `class` attribute to add elements to a class.
- Use a `.` between the element name and the class name to select a specific element in that class.
- Use `.classname` to select any elements that belong to the class.
- You can specify that an element belongs to more than one class by placing multiple class names in the `class` attribute with spaces between the names.
- You can validate your CSS using the W3C validator, at <http://jigsaw.w3.org/css-validator>.



HTMLcross

Here are some clues with mental twist and turns that will help you burn alternative routes to CSS right into your brain!



Across

3. Styles are defined in these.
5. Selects an element.
8. Each rule defines a set of properties and _____.
10. Defines a group of elements.
11. Property that represents font color.
13. Ornamental part of some fonts.
14. How elements get properties from their parents.
15. Property for font type.

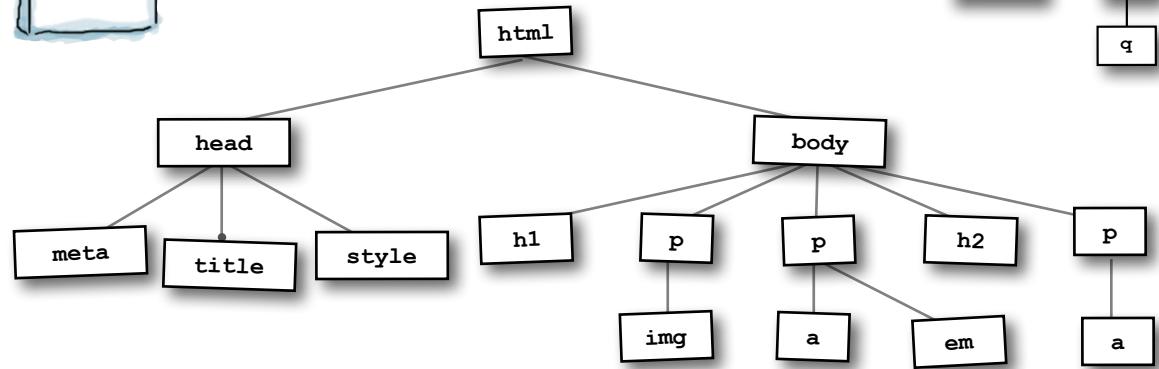
Down

1. Fonts without serifs.
2. You can place your CSS inside these tags in an HTML file.
4. An external style file is called this.
6. With inheritance, a property set on one element is also passed down to its _____.
7. Won this time because they used external stylesheets.
9. They really wanted some style.
12. Use this element to include an external stylesheet.

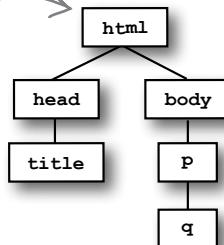


Markup Magnets Solution

Remember drawing the hierarchy diagram of HTML elements in Chapter 3? You did that again for the Lounge's main page. Here's our solution.



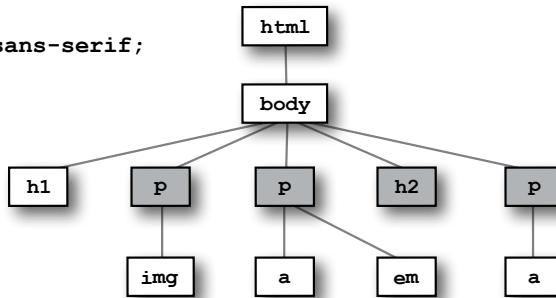
Like this



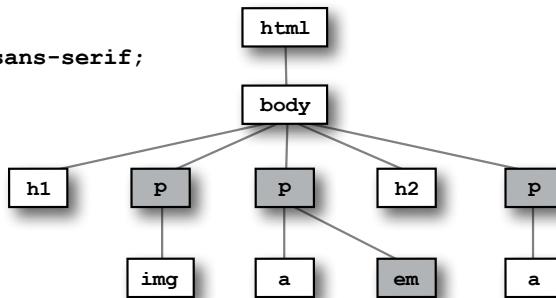
Sharpen your pencil Solution

The selected elements are colored:

```
p, h2 {
  font-family: sans-serif;
}
```



```
p, em {
  font-family: sans-serif;
}
```





Sharpen your pencil

Solution

Your turn: add two classes, "raspberry" and "blueberry" to the correct paragraphs in "elixir.html" and then write the styles to color the text blue and purple, respectively. The property value for raspberry is "blue" and for blueberry is "purple".

```
body {
    font-family: sans-serif;
}

h1, h2 {
    color: gray;
}

h1 {
    border-bottom: 1px solid black;
}

p {
    color: maroon;
}

p.greentea {
    color: green;
}

p.raspberry {
    color: blue;
}

p.blueberry {
    color: purple;
}
```

Head First Lounge Elixirs

file:///chapter7/lounge/beverages/elixir.html

Our Elixirs

Green Tea Cooler



Chock full of vitamins and minerals, this elixir combines the healthful benefits of green tea with a twist of chamomile blossoms and ginger root.

Raspberry Ice Concentration



Combining raspberry juice with lemon grass, citrus peel and rosehips, this icy drink will make your mind feel clear and crisp.

Blueberry Bliss Elixir



Blueberries and cherry essence mixed into a base of elderflower herb tea will put you in a relaxed state of bliss in no time.

Cranberry Antioxidant Blast



Wake up to the flavors of cranberry and hibiscus in this vitamin C rich elixir.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Head First Lounge Elixirs</title>
    <link type="text/css" rel="stylesheet" href="../lounge.css">
  </head>
  <body>
    <h1>Our Elixirs</h1>
    <h2>Green Tea Cooler</h2>
    <p class="greentea">
      
      Chock full of vitamins and minerals, this elixir
      combines the healthful benefits of green tea with
      a twist of chamomile blossoms and ginger root.
    </p>
    <h2>Raspberry Ice Concentration</h2>
    <p class="raspberry">
      
      Combining raspberry juice with lemon grass,
      citrus peel and rosehips, this icy drink
      will make your mind feel clear and crisp.
    </p>
    <h2>Blueberry Bliss Elixir</h2>
    <p class="blueberry">
      
      Blueberries and cherry essence mixed into a base
      of elderflower herb tea will put you in a relaxed
      state of bliss in no time.
    </p>
    <h2>Cranberry Antioxidant Blast</h2>
    <p>
      
      Wake up to the flavors of cranberry and hibiscus
      in this vitamin C rich elixir.
    </p>
  </body>
</html>
```



Exercise Solution

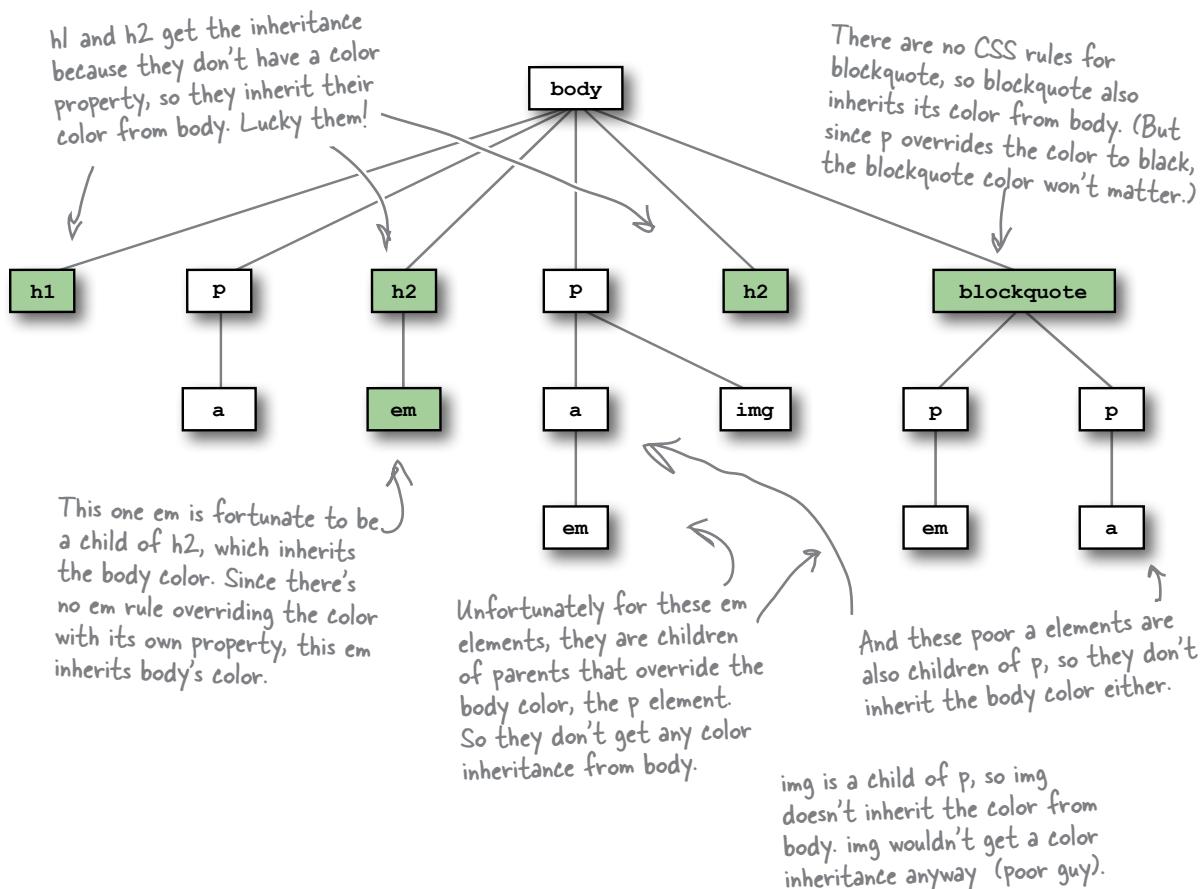
Who gets the inheritance?

Sniff, sniff; the `<body>` element has gone to that great browser in the sky. But he left behind a lot of descendants and a big inheritance of color “green”. Below, you’ll find his family tree. Mark all the descendants that inherit the `<body>` element’s color green. Don’t forget to look at the CSS below first.

Here’s our solution:

```
body {
  color: green;
}

p {
  color: black;
}
```





BE the Browser Solution

Below, you'll find the CSS file with some errors in it. Your job was to play like you're the browser and locate all the errors. Did you find them all?

```

<style>
  body {
    background-color: white
    h1 {
      color: gray;
      font-family: sans-serif;
    }
  }
  h2, p {
    color:
  }
  <em> {
    font-style: italic;
  }
</style>

```

<style> No HTML in your CSS! The **<style>** tags are HTML and don't work in a CSS stylesheet.

body { Missing semicolon

background-color: white Missing }

h1 { Extra comma

color: Missing property name and colon

h2, p { Missing property value and semicolon

** {** Using the HTML tag instead of just the element name. This should be em.

No **</style>** tags needed in the CSS



In your “elixir.html” file, change the greentea paragraph to include all the classes, like this:

```
<p class="greentea raspberry blueberry">
```

Save and reload. What color is the Green Tea Cooler paragraph now?

purple

It's purple because the blueberry rule is last in the CSS file.

Next, reorder the classes in your HTML:

```
<p class="raspberry blueberry greentea">
```

Save and reload. What color is the Green Tea Cooler paragraph now?

purple

It's still purple because the ordering of the names in the class attribute doesn't matter.

Next, open your CSS file and move the p.greentea rule to the bottom of the file.

Save and reload. What color is the Green Tea Cooler paragraph now?

green

Now, it's green, because the greentea rule comes last in the CSS file.

Finally, move the p.raspberry rule to the bottom of the file.

Save and reload. What color is the Green Tea Cooler paragraph now?

blue

Now, it's blue, because the raspberry rule comes last in the CSS file.

After you've finished, rewrite the green tea element to look like it did originally:

```
<p class="greentea">
```

Save and reload. What color is the Green Tea Cooler paragraph now?

green

Okay, now the <p> element only belongs to one class, so we use the most specific rule, which is p.greentea.



HTMLcross Solution

